TECHNICAL DOCUMENT 3199
December 2005

ROBART I: In Retrospect

H. R. Everett

SSC San Diego

TECHNICAL DOCUMENT 3199
December 2005

ROBART I: In Retrospect

H. R. Everett

SSC San Diego
San Diego, CA 92152-5001

F. D. Unetic, CAPT, USN
Commanding Officer

C. A. Keeney
Executive Director

ADMINISTRATIVE INFORMATION

The work described in this report was performed for the Defense Advanced Research Projects Agency (DARPA) by the Chief of Robotics (Code 23705) of the Advanced Systems and Sciences Division (Code 237) Space and Naval Warfare Systems Center San Diego.

Released under authority of
M. Machniak, Head Advanced
Systems and Sciences Division

CONTENTS

Figures

Tables

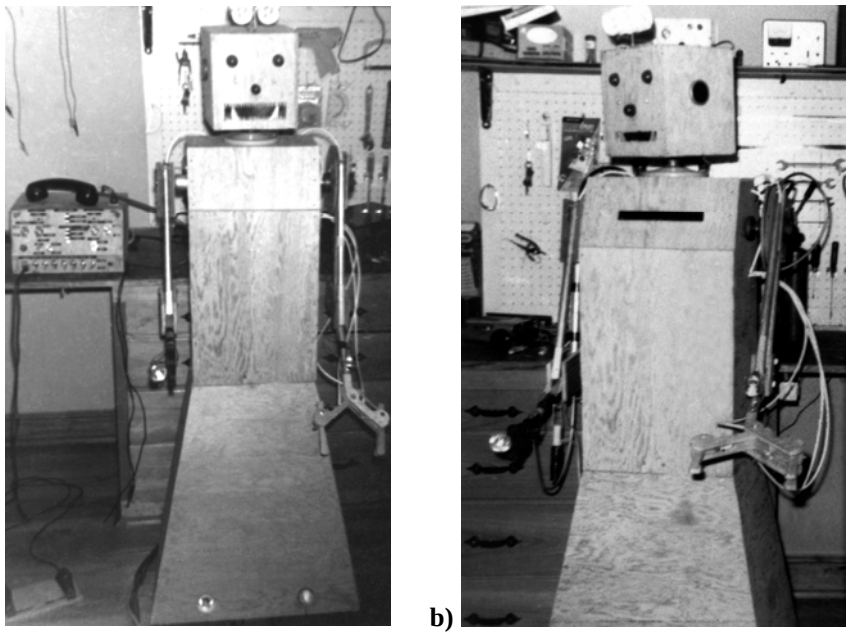
INTRODUCTION

The following introductory sections provide a brief overview of a few robots that preceded and hence influenced the *ROBART Series*. It is somewhat amusing to note the advancements since made in the supporting technologies.

WALTER (1965-1966)

WALTER (Figure 1) was a 5-foot-tall semi-anthropomorphic robot I constructed during my sophomore year in high school as a science fair entry. Strictly a teleoperated system with no onboard intelligence, *WALTER* was capable of forward or reverse travel, using two 8-inch drive wheels made of ¾-inch plywood and a pair of 2-inch roller-skate wheels in front for steering. The steering mechanism was solenoid-actuated under *bang-bang* control, with a spring-loaded center default position. A 20-foot umbilical tether supplied 117-volt AC power from the control station shown on the left side of Figure 1a.

The right arm was driven by linear actuators constructed from ¼-inch threaded rod for elbow and shoulder movement, powered by a sewing machine motor and a kitchen mixer, respectively. The left arm had only a single degree of freedom at the elbow (I ran out of motors). Its associated actuator was coupled to the prime mover from an old movie projector. All the motors were single-speed, series-wound universal type, controlled (by onboard relays) from the remote operator console. The linear actuators were linked to their respective joints by tendons made from bicycle hand-brake cables.



a) **b)**

Figure 1. a) *WALTER* (circa 1965) was a teleoperated anthropomorphic robot constructed as a high-school science-fair entry; **b)** *WALTER's* grippers, elbows, and shoulder joints were tendon-actuated.

The left and right grippers were also different, but similar in that they both lacked wrist movement. The right gripper was fashioned from a 10-inch fuse puller, aligned for

grasping objects oriented in the horizontal plane. The left gripper was somewhat more complex, constructed from ¼-inch hardwood with two degrees of freedom as illustrated in Figure 1b, and oriented to grasp vertical objects. All gripper joints were tendon-driven by cables that were spring-coupled to powerful solenoids removed from innumerable washing machines.

WALTER's head could pan left or right approximately 45 degrees, driven through tendons by a linear actuator mounted in the base to keep the center of gravity low. Load-bearing joints (head pan axis, shoulder, elbows) were fashioned from ball-bearing roller-skate wheels. A photocell was mounted on top of the head to monitor ambient light conditions, and, of course, the obligatory flashing lamps served as eyes and nose. Two microphone ears and a speaker behind the mouth opening provided for remote communications via the telephone handset shown in Figure 1a.

The robot and control-console electronics were vacuum-tube based. One interesting submodule was a capacity-operated relay coupled to a touch sensor in the right gripper. The sole purpose of this circuitry was to discourage pulling and prodding by curious onlookers; any stray finger that poked its way into the open claw would be met by a startling and decidedly effective warning snip. The resounding thump of the actuating solenoid very effectively accentuated the message.

WALTER met his demise one day in 1967 at the hands of our cleaning lady (bless her heart). I had been experimenting with some six-transistor portable radios that sold at the time for around 5 dollars apiece, trying to come up with a low-cost RF control scheme. The idea was to tune each of the four receivers to unused portions of the AM band, and use a continuous-tone transmitter that could be switched to any one of these four frequencies. Half-wave rectifiers attached to the audio outputs of the individual radios energized sensitive meter relays that controlled the forward, reverse, left, and right power relays in the drive circuitry.

As fate would have it, the unsuspecting maid timidly entered the confines of my bedroom workshop one day when I was not at home and turned on the ancient pre-World-War-II *Lewyt* vacuum cleaner my dad had rebuilt six times just in my brief lifetime. The motor brushes had long since worn down to their springs, which arced across the pitted commutator segments with such intensity that all TV and radio reception for two blocks was blanked out whenever the machine was running. *WALTER's* radios responded instantly to this rich broad-band interference, randomly applying power in a mindless fashion to drive motors and steering solenoids alike. The robot lurched forward, twisting and turning, motors whining and solenoids clacking, only to be immediately decapitated with one mighty swing of a *Lewyt* rug sweeper. When I got home the vacuum was still running, poor *WALTER* was a total loss, the front door was swinging on its hinges, and the skittish maid had vanished, never to return.

CRAWLER I **(1966-1968)**

I had been bitten by the bug, it seemed, and was now fascinated with the idea of building a free-roaming robot unencumbered by any sort of tether. There was little point

in trying to refurbish *WALTER*; structural damage notwithstanding, all the electrical components were rated for 117 volts AC. My next creation had to be battery powered, so I began to amass an impressive collection of DC motors, relays, and other diverse components while sorting out the design in my head. The end result was *CRAWLER I* (Figure 2), intended to be my junior-year science project, but the eagerly anticipated event was unfortunately canceled due to faculty indifference.

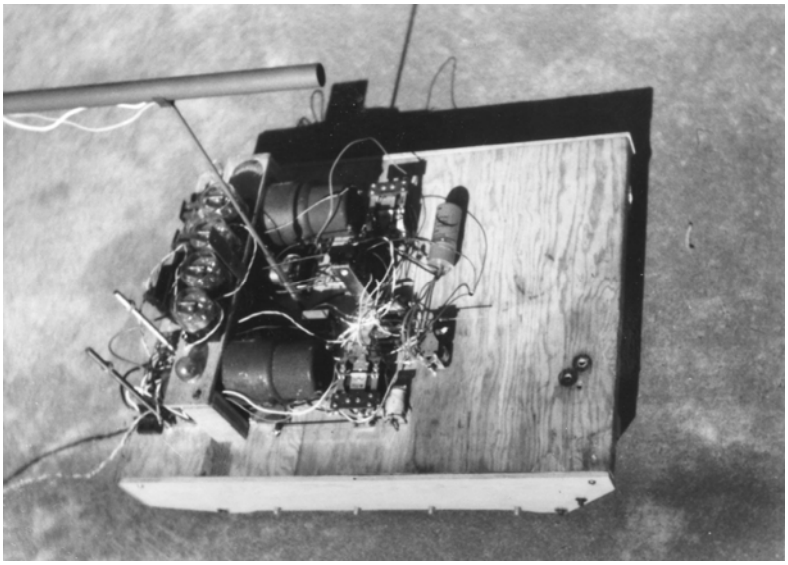


Figure 2. Photo of the wooden *CRAWLER I* (circa 1966) in early stages of development.

I had also decided to build a tracked vehicle for improved maneuverability. A pair of 24-volt DC gearmotors from a aircraft surplus catalog were mounted on a 18- by 13-inch plywood base (Figure 4), driving left and right tracks fashioned from 1.5-inch rubber timing belts turned inside out. Control was again provided by electromechanical relays, but each motor had a centrifugal speed-limiting switch that could be adjusted to achieve precise straight-line travel. By adding an override circuit on the stationary side of the slip rings that fed the centrifugal governor, it was possible to momentarily boost the motor rpm to maximum. *Skid steering* was achieved by providing differential speed commands in this fashion or by stopping one motor altogether. The vehicle could also turn in place by reversing one track.

The tough part in building an autonomous vehicle, of course, lies in how to control its motion, made even tougher in an era that predated microprocessors and low-cost sensors. I had in mind a platform that would drive around until it encountered an object, then alter course in an intelligent fashion. I also wanted it to automatically recharge the onboard lead-acid motorcycle batteries when they ran low. Like most engineers, I tackled the tougher issue first: automatic recharging. I settled on a beacon homing scheme with an ordinary 60-watt light bulb as the source. The scanning sensor was simply a cadmium-sulfide photoresistor mounted in the end of a 12-inch plastic tube, which was rotated in the horizontal plane of the beacon by a small DC gearmotor from a sign display (Figure 3). A special slip-ring sensor at the bottom of the rotating shaft (Figure 4) was used to index the 0-degree position coinciding with the forward axis of travel.

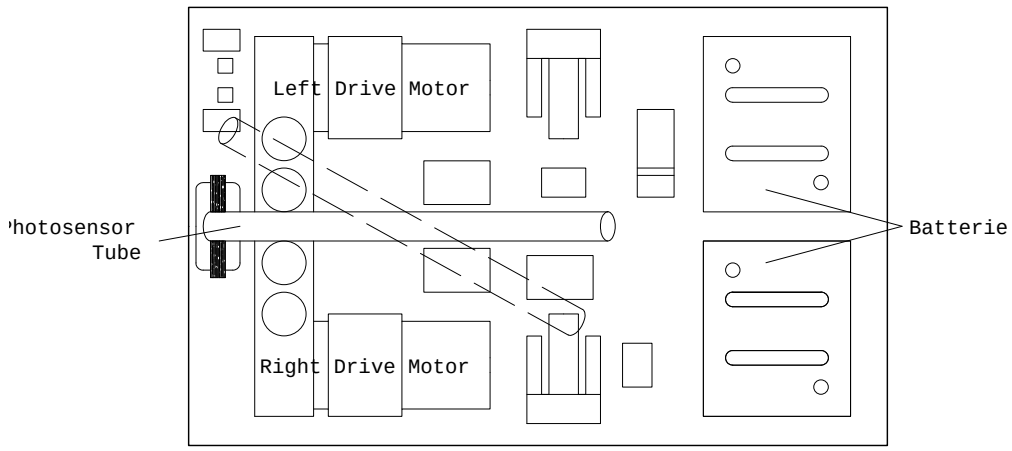


Figure 3. Top view of the component layout on *CRAWLER I*, showing the photosensor collimating tube aligned with the forward axis of travel, and again displaced 30 degrees.

All the control logic was implemented with surplus mechanical relays. The photocell scanner rotated at a constant rate, with vehicle turning initiated by the shaft index sensor and halted by beacon detection (or vice versa). The first sweep was used to determine if the beacon was present, and if so, in which direction the vehicle needed to turn. A pair of triple-pole, double-throw relays was used to implement an electromechanical flip-flop to remember on which side the beacon had most recently been seen. *CRAWLER I* was therefore not only my first autonomous robot, but also the first to have an actual memory, albeit but a single bit! (Keep in mind this was a high-school science project back in the technology-starved mid-sixties.)

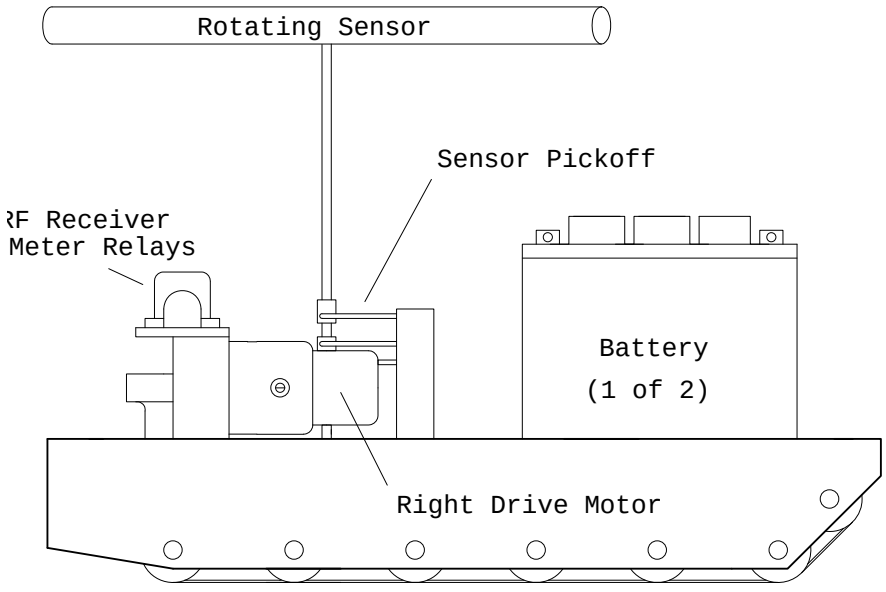


Figure 4. A rotating photocell sensor was used on *CRAWLER I* to locate and track a homing beacon for automatic recharging.

For example, if the beacon lay off to the right side, the scanner would initiate a platform turn in that direction by stopping the right drive motor every time the collimating tube passed through the forward index (i.e., pointed straight ahead). As the sweep continued from left to right, the photocell would eventually detect the beacon,

whereupon the control logic would restart the right drive motor. This process would repeat with every sweep. Each time the turning action would last for a shorter period than before, because the beacon would have moved closer to the 0-degree index as the robot turned toward it. When the beacon lay dead ahead, the sweep would start and then immediately stop the turn, resulting in straight-line travel. An early schematic diagram (reproduced from my original notes) of this beacon-tracking scheme is shown in Figure 5.

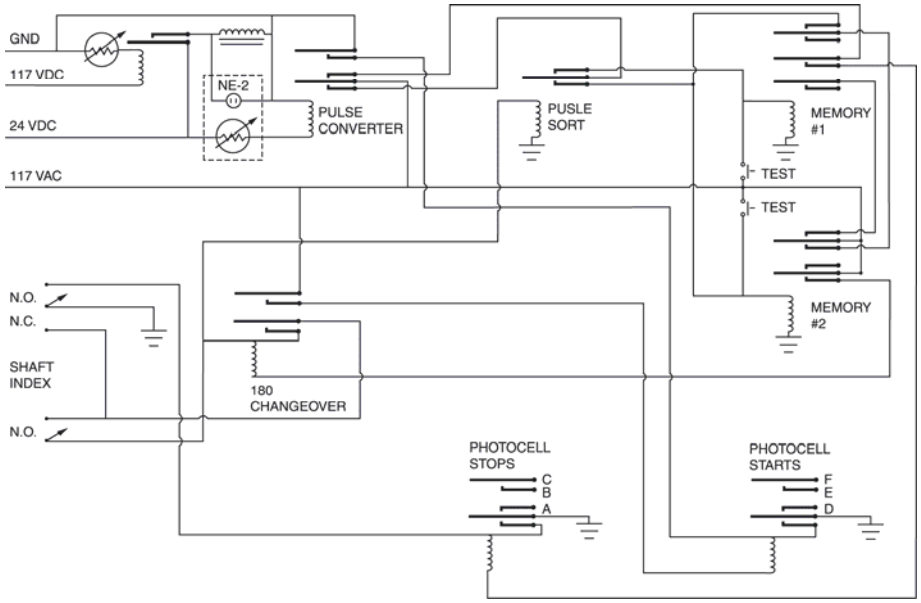


Figure 5. Schematic diagram of the first prototype beacon-tracking system used on *CRAWLER I* before conversion to 12 volts DC. Left and right drive motor connections are designated A-B-C and D-E-F.

A minor wrinkle in this approach was that if the beacon happened to be off to the left side versus the right, the events that initiated and terminated the turning action had to be interchanged. In other words, beacon detection would start the turn while the sweep index would halt it. The mechanics of this strategy were worked out in the relay logic, relying on the *photocell-left-or-right* information stored in the flip-flop to control the process. Fine tuning to achieve stability was accomplished by varying the sweep speed of the scanner with a rheostat. The results were surprisingly effective.

Providing for truly autonomous operation meant adding some type of collision-avoidance sensor and implementing an associated scheme of intelligent reaction. In 1966, tactile sensing was about the only practical means available (due to limitations in the technology and also my budget), and so home-made feelers were subsequently installed on the four corners of the platform to support this task. The first implementation consisted of a short length of guitar string extended through the center of a small screw-eye; any deflection of the wire caused contact closure, completing a simple normally open circuit. Similar implementations are reported by Russell (1984), Schiebel (1986), and Brooks (1989). An enhanced version of this sensor (Figure 6) involved looping the wire back on itself through a second screw-eye to form a circle, thus widening the protected area.

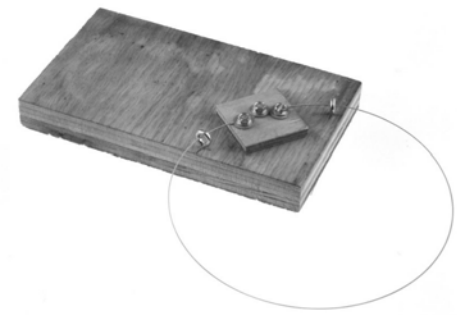


Figure 6. Tactile sensors situated at the four corners of the *CRAWLER* robots were fabricated from guitar strings looped through the center of a pair of small screw-eyes.

Intelligent response was another matter, as single-chip microcontrollers were not yet even a figment of anyone's imagination in those days. My Hollywood-inspired image of a computer was centered around a lot of flashing lights and punched cards. I had already wired dozens of very impressive indicator lamps in parallel with the relay coils of the *CRAWLER's* logic and control circuitry (for diagnostic purposes, of course). Operating the *CRAWLER* with the four-channel RF transmitter developed on *WALTER* had quickly become boring, so it seemed the appropriate thing to do was to build a punched-card reader.

The robot's surrounding environment could be simplistically described by four state variables associated with the tactile sensors situated at each of the platform corners. By comparing these sensor input states to a four-bit address field punched into each card, the correct response to any particular scenario could be read from the output section of the card with an address matching the input conditions. The robot would simply stop whenever input conditions changed state and cycle the cards until finding a match, then read the preprogrammed response (i.e., drive and steering commands). I was really excited about the prospect of building this card reader and made fair progress using modified index cards with eight photocells to detect holes from a standard office punch. An actual 3.5- by 8-inch card is shown in Figure 7. The top row of holes represented the inputs, while the bottom row controlled the outputs. The centered hole at the end was intended to verify proper orientation but later found unnecessary.

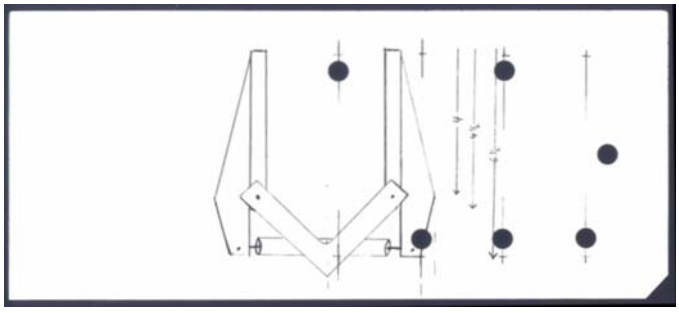


Figure 7. This actual punched card used on *CRAWLER I* shows the two rows of holes representing input and output data. The sketch on the back is a preliminary gripper design (for *CRAWLER II*) that was abandoned in favor of the vise-grip implementation shown later in Figure 10.

A partial schematic (again reproduced from my notes) of the original card reader is shown in Figure 8 below. The individual illumination sources for the eight opposing photocells were 3-volt pilot lamps, wired in series to ensure the entire string would extinguish to prevent faulty readings if any single bulb burned out. The lamps were powered by the 12-volt battery at half their rated filament voltage to ensure extended life, and the reduced light output prevented premature activation of the photodetectors through the thin index-card paper. But the mechanics associated with reliably recycling the stack of cards (once all had been read) proved too much for my limited shop facilities, so I resorted to using a 12-inch circular disk of poster paper traced from a 33-rpm record album.

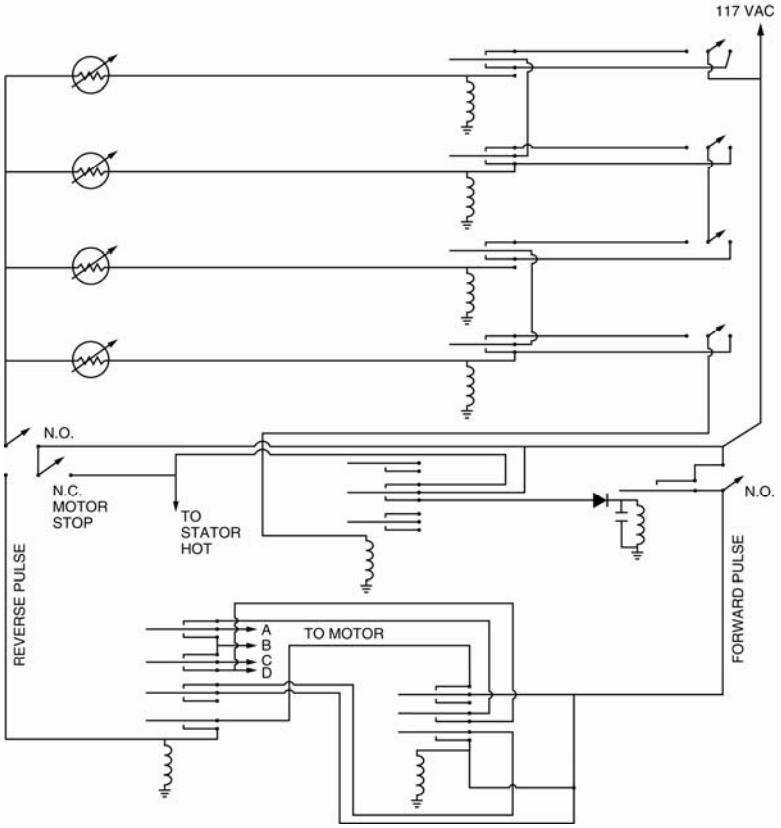


Figure 8. Partial schematic diagram (input holes only) of the original (AC-powered) punched-card reader for *CRAWLER I*, circa 1966. The four single-pole-double-throw switches at upper right represent the corner tactile sensors. In the follow-on 12-volt DC version, these were implemented as relays triggered by the guitar-spring sensors shown in Figure 6.

This approach greatly simplified matters. The address and output fields were aligned along the radial axis of the disk with 16 possible states as shown in Figure 9, with the most significant bit towards the outer periphery. The disk would rotate at 6 rpm while the photocells looked for a hole pattern corresponding to the sensor input states. When a match was found, the drive motor was disabled and the output field would be read from the stationary disk, thus determining the desired relay states for left and right track drive and direction. The output holes were punched in radial columns offset 78.75 degrees from their associated input columns to allow sufficient room for the two photocell arrays. The circular card was secured to a rubber-covered drive capstan with a ¼-inch wingnut and washer.

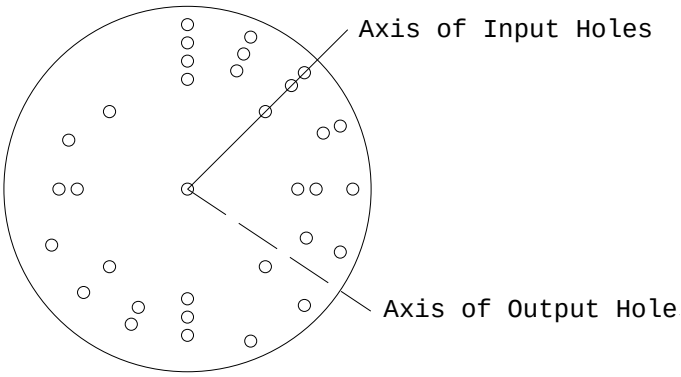


Figure 9. Mechanical problems with the stacked-card transport mechanism forced a switch to the circular card format shown above. Punched output holes (not shown) were inserted between the input address fields.

CRAWLER II (1968-1971)

All the added features (particularly the 12-inch disk reader) necessitated a complete repackaging of the *CRAWLER's* mechanical layout, so I scrapped the plywood prototype altogether and built an aluminum chassis. The result was *CRAWLER II*, basically the same size, but with the electronics implemented in a layered approach, and equipped with a hydraulic arm and gripper configuration as illustrated in Figure 10.

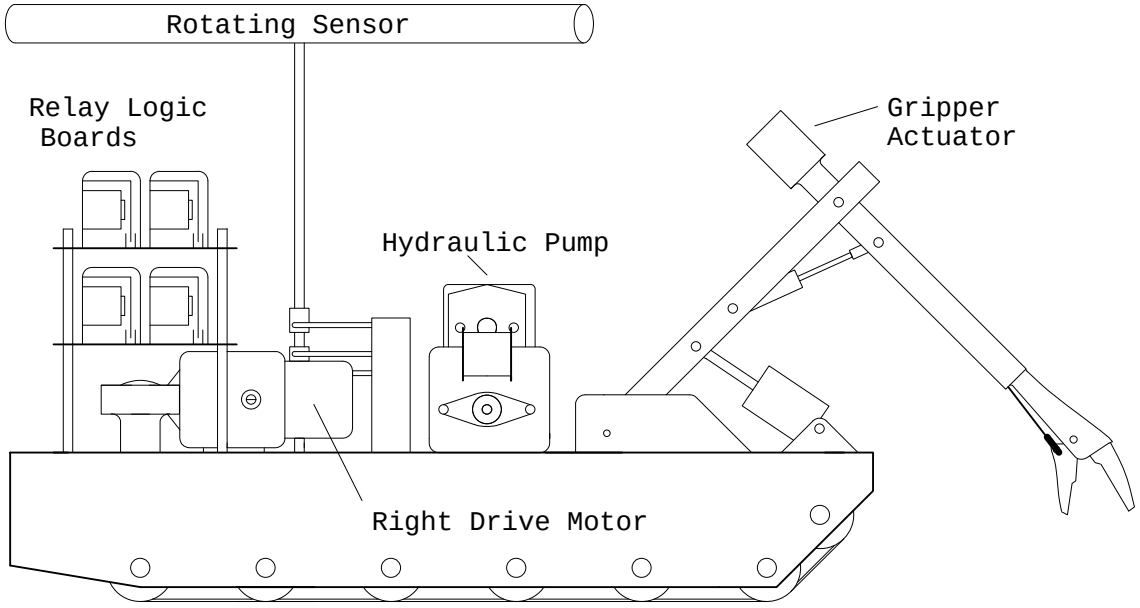


Figure 10. *CRAWLER II* (shown without the circular disk reader) was a teleoperated platform equipped with a two-degree-of-freedom hydraulic arm.

I had begun experimenting earlier with some miniature hydraulic cylinders fashioned by encapsulating 30- and 50-cc irrigation syringes inside copper-tubing sheaths with epoxy glue. Considerable force could be generated with one of these devices when operated at about 100 psi; solenoid valves from discarded washing machines were modified to provide flow control. A surplus chemical-injection pump was used to pressurize an accumulator made from a 4-inch length of 3-inch-diameter copper pipe,

capped on both ends. The gripper force was quite powerful. While attempting to explore the limits of remote-operator dexterity, I once squeezed the locomotive of my brother's train set just a wee bit too hard, rendering it no longer compatible with HO gauge track.

An early prototype of the hydraulic system is shown in Figure 11 below. The cylindrical tank in the lower-left corner served as the high-pressure accumulator. Above it can be seen the supply and return solenoid valves (solenoid coils are removed). The hydraulic supply reservoir is the horizontal tank on the upper-right corner of the plywood base, with the pump drive motor immediately to its left. The motor was coupled through a spur gear to a worm-gear reduction unit driving an eccentric that produced linear motion for the pump. The temporary pump assembly was implemented as a long brass cylinder shown at the very bottom of the plywood base. The motor, reduction gears, and pump shown in this photo were later replaced by an integrated off-the-shelf product marketed as a chemical-injection unit for an air-conditioning cooling tower. The second-generation manipulator sketch shown earlier in Figure 7 was intended to replace the crude wooden prototype partially shown in the lower-left corner of the photo, but was abandoned in favor of the vise-grip implementation depicted in Figure 10.

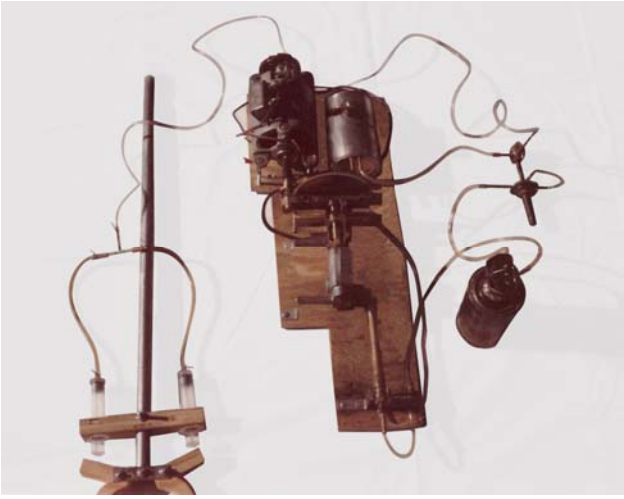


Figure 11. An initial test prototype of the hydraulic system is shown in this 1967 Polaroid® photo. The 30-cc irrigation syringes that actuate the manipulator (lower left) were not yet encapsulated in copper pipe.

Unfortunately, the bulky disk reader and the final manipulator design would not both fit on the platform at the same time, and the modified hydraulic components were all rated for 117 volts AC. In addition, there was essentially no way to control the manipulator in an autonomous fashion, so *CRAWLER II* had to revert back to tethered control. The few photographs I have of *CRAWLER I* were taken by one of my high school buddies who owned a Polaroid® camera; since most of the *CRAWLER II* development was performed while I was away at college, I regrettably do not have any pictures. Work on *CRAWLER II* ceased my junior year, when I cannibalized the onboard logic control unit to automate our very mechanized homecoming display at Georgia Tech.

A-4 SKYHAWK

I constructed this 1/3-scale model of the McDonnell-Douglas *A-4 Skyhawk,* flown at the time by the Blue Angels precision flight demonstration team, in the late seventies while stationed with the Navy in Montgomery, AL. I had initially intended to build a go-cart for my 4-year-old son, but we could only find three lawn-mower wheels in my shop, so we opted to build an airplane instead. It started off as a rather crude triangle of plywood with a seat up front, but our creative juices got to flowing and we kept thinking of ways to make it better and more realistic (Figure 12). While clearly not a robot, this project did substantially influence my later work on the *ROBART Series*, particularly with respect to fiberglass prototyping.

Figure 12. We first embellished by adding a three-dimensional fuselage structure atop the triangular wing, and closed in the cockpit and nose section as shown in ¼-inch plywood.

We then installed the empennage with its horizontal and vertical stabilizers, complete with moveable flight surfaces (rudder and elevator) cabled directly to the rudder pedals and control stick in the cockpit. The ailerons on the trailing edges of the wings were similarly actuated, but the flaps immediately below them were motorized. Two 12-volt car batteries provided the necessary power. Once we had electrical options, there was seemingly no end to the possibilities, such as the obvious cockpit, landing, collision avoidance, and tail lights, followed by a linear actuator for raising and lowering the tailhook mechanism. Twin gun barrels were then installed, in close to the fuselage on the leading edges of both wings, their internal strobe lights synchronized to a motor-driven mechanical noise-maker controlled by the trigger switch on the control stick.

Next came a rather impressive engine simulator, fashioned from a modified electromechanical siren and a hair-dryer blower, instrumented with microphones that fed a 12-watt stereo amplifier. The amplifier speakers were situated so the turbine whine produced by the siren appeared to come from the engine intakes, while the roar of simulated jet exhaust generated by the blower mike poured forth from the tailpipe. The respective speeds of the siren and blower, as well as the amplifier volume, were

controlled by the engine throttle in the cockpit. The characteristic inertial lag of turbine response was fairly realistically simulated by a series of thermal-delay relays.

Figure 13. Front-quarter view during the early stages of fiberglassing. The landing light is visible under the starboard wing, and simulated bomb loads hang in the racks just inboard of the mainmounts, but the *Sidewinder* missiles have not yet been installed.

Once the basic actuators and associated wiring were completed, we began building up the body and wing surfaces with fiberglass, using a tennis ball as a form for the nose, short lengths of garden hose to define the air intake cowlings, and poster paper to create the rounded portions of the fuselage. For extra stiffness, the upper wing surface was built up of two layers of fiberglass over a ¼-inch plywood skin, which in turn was glued down on an internal rib structure to achieve the desired airfoil contour. This configuration allowed the wing assembly to easily support the envisioned loading to be imposed by a never-ending progression of climbing admirers and future naval aviators.

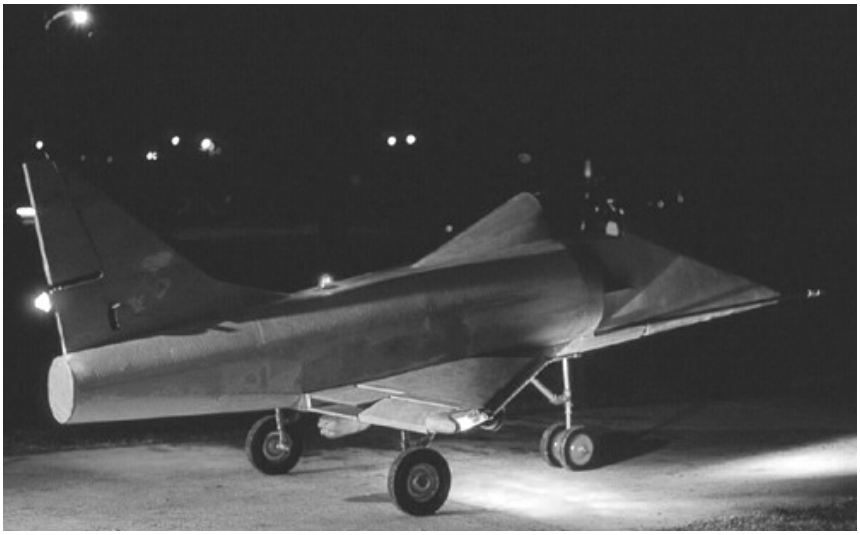


Figure 14. This hormone-inspired nighttime test of the cockpit, landing, navigation, and anti-collision lights was conducted with the simulated engine at full power, of course, just to convince the neighbors that we had really lost our minds.

I talked the Blue Angels into painting the finished product, shown in Figure 15 next to the real thing at NAS Pensacola, so we could use it as an officer recruiting aid for aviation programs.[1] Note the landing gear has been lengthened to yield the correct elevation and attitude, and sports additional detail such as simulated oleos, brake lines, a nosewheel-steering linkage, and a weight-on-wheels switch. The extended height, however, required incorporating dual tires on the mainmounts for added stability. The Blue Angels' maintenance personnel were so taken with the working functionality that I received a box of additional "doo-dads" from their spares to enhance the realism even further.

Figure 15. Once painted, the surface finish was smooth and shiny enough to where you could see your own reflection. Note the working tailhook below the rear fuselage, and the semi-functional instrument panel just visible in the cockpit.

I left the completed model with the Navy Recruiting District, Montgomery, AL, when I was transferred as an Engineering Duty Officer in 1980 to the Naval Postgraduate School in Monterey, CA. Its whereabouts today is unknown.

[1] See a copy of the article "Which One's One?" in the April, 1980 issue of *Naval Aviation News*, at: http://www.blueangels.org/History/NANews/EdCor/Apr80/Apr80.htm.

1
System Overview

"The human brain starts working the moment you are born and never stops until you stand up to speak in public."

George Jessel

ROBART I was my thesis project at the Naval Postgraduate School in Monterey, CA, during the early eighties (Everett, 1982). My fledgling career as a naval officer kept me pretty busy after graduation from Georgia Tech, leaving little time for further experimentation in robotics. Acceptance to graduate school looked like a perfect opportunity to re-engage after a 7-year hiatus, especially if I could kill two birds with one stone by satisfying my thesis requirement in the process. I was so excited by this prospect that I began construction even before classes formally convened in the fall of 1980. While not the first autonomous robot I had built, it was nonetheless my first computer-controlled version. Since my undergraduate degree had been in electrical, I had decided to pursue a masters in mechanical engineering.

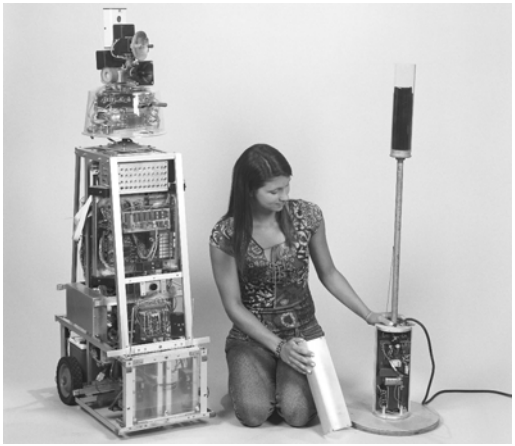


Figure 1-1. Office of Naval Research intern Lisa Dakis poses with *ROBART I* and its battery charging station in this 2005 file photo taken at the Space and Naval Warfare Systems Center San Diego.

There were not a lot of other mobile robots in those days from which to glean valuable lessons learned, which in retrospect may have been somewhat of a good thing. Probably the most famous example of earlier work was *Shakey*, built at the Stanford Research Institute (now SRI International) in the mid-sixties to early-seventies (Nilson, 1984), and

just recently inducted into the *Robot Hall of Fame* at Carnegie-Mellon University (CMU, 2004). *Shakey* was developed from a decidedly artificial-intelligence (AI) perspective, in that it was a vision-based mobile platform linked to a remote mainframe computer, which in turn laboriously analyzed the video imagery and planned appropriate actions. To facilitate this image processing, the robot performed in a highly structured laboratory setting of neutral backgrounds and colored shapes. The research emphasis was offline task decomposition and rule-based planning to achieve a desired goal, and execution was rather time-consuming. It typically took 6 to 8 hours, for example, to find and push a certain colored block to some new position, perhaps moving other blocks out of the way as needed (Brooks, 2002).

Another pioneering effort that could have influenced me, had I been aware of it at the time, was Hans Moravec's impressive work with the *Stanford Cart*, conducted from 1973 to 1980 at the Stanford University Artificial Intelligence Laboratory. The *Stanford Cart* was similar in many respects to *Shakey*, in that it was a video-equipped remote platform linked to a mainframe computer that processed the image data and planned accordingly. The *Cart* differed from the earlier SRI robot in at least two rather significant areas: (1) it employed stereo as opposed to monocular vision, which provided valuable range information, and (2) it operated in less structured environments, with more emphasis on collision avoidance. Moravec (1983) provides the following candid assessment:

> "The system was reliable for short runs, but slow. The Cart moved one meter every ten to fifteen minutes, in lurches. After rolling a meter it stopped, took some pictures and thought about them for a long time. Then it planned a new path, executed a little of it, and paused again. It successfully drove the cart through several 20 meter courses (each taking about five hours) complex enough to necessitate three or four avoiding swerves; it failed in other trials in revealing ways."

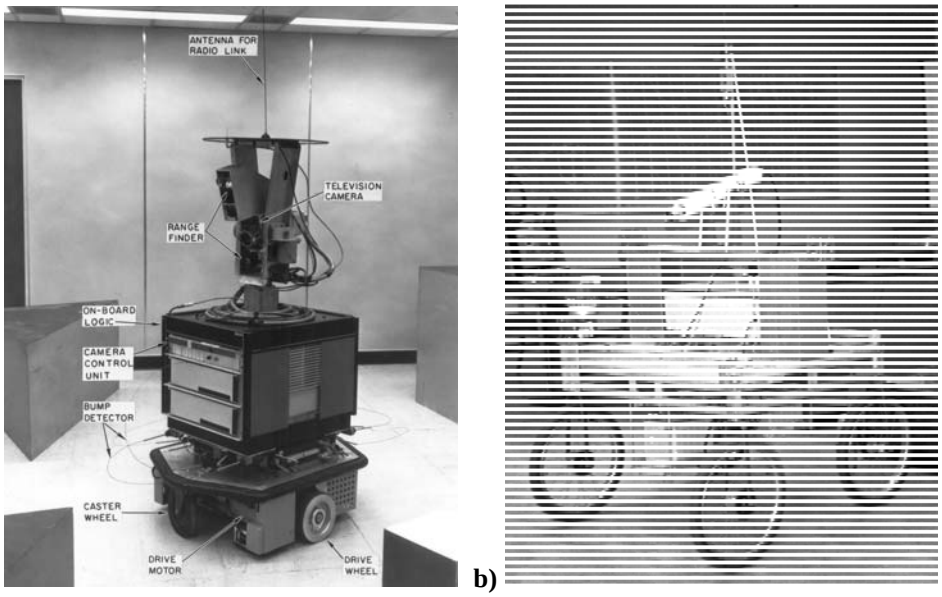


Figure 1-2. **a)** SRI's *Shakey* employed a monocular vision system to plan subsequent actions in a structured laboratory setting (courtesy SRI); **b)** the *Stanford Cart* later used "sliding camera" stereo-vision to support collision avoidance in more unstructured environments (courtesy Hans Moravec).

I actually made a field trip to Stanford University with Professor William Culbreath of the Naval Postgraduate School sometime in late 1981 or early 1982, but approximately 10 years had passed since the work on *Shakey* ended. Very little was ever published on this project (i.e., Nils Nilson's technical report entitled *Shakey the Robot* (Nilson, 1984) was not released until 2 years after I finished my thesis work and had left Monterey). Furthermore, since SRI had spun off from Stanford University during the Vietnam War, we were looking in the wrong place anyway. It was a lot harder in those days to track down related research efforts in general, since such projects were scarce, there was no plethora of mobile robot conferences as we have today, or any world wide web to search. We wound up talking to Urs Ulsasser about an omnidirectional platform (Figure 1-2), but got no insights into the *Shakey* project, or even pointers to Moravec's work in the university's AI Lab.

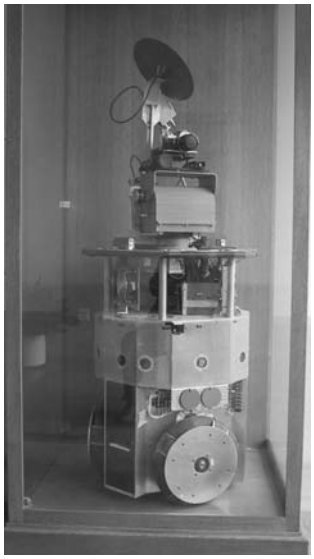
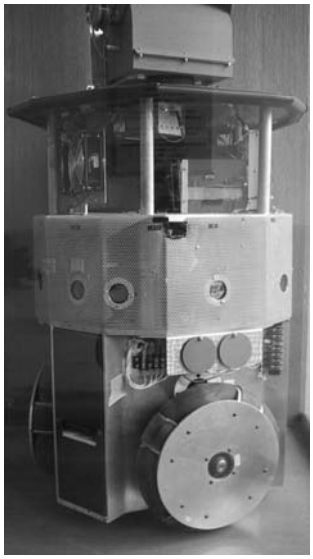
a)  **b)** 

Figure 1-3. a) The Stanford University robot *Mobie* on display; **b)** close up of *Mobie's* omnidirectional drive wheel (courtesy Stanford University).

Now, obviously I did not have ready access to a highly structured lab setting, a mainframe computer, or sophisticated image-processing algorithms. Nor did I have any established background in classical artificial intelligence. I was more interested in a robot that could operate in real time in a real-world environment, like my earlier *CRAWLER* series, which had functioned reasonably well despite extremely primitive relay logic. Surely the availability of an onboard eight-bit microprocessor would allow some monumental improvements in performance relative to a handful of electro-mechanical relays! So armed with perhaps an over-abundance of naïve optimism, I began what was to become a 25-year-plus quest for machine intelligence, based on simple underlying reactionary behaviors with evolutionary upgrade capability.

1.1 Design Considerations

Since a mobile platform must move about in the performance of its duties, a robust ability to navigate and avoid collisions with surrounding objects is crucial. This is perhaps the biggest design concern, and involves careful selection and placement of sensors as well as effective data utilization within the software. Collision-avoidance routines must be set up to maximize the use of all available information, and take into

account impending collisions that arise as a result of previous evasive action. The robot cannot blindly back up some pre-specified distance to the left in response to a front-right-bumper impact, for example, or it may wind up hitting another object to the rear.

Once the capability to safely maneuver has been addressed, the machine's ability to accomplish its mission becomes the dominant issue. For *ROBART I*, the objective was to serve as an autonomous security guard in a typical home environment. Significant attention was therefore paid to various methods of intrusion detection, as well as some means to sense the presence of smoke, fire, toxic gas, flooding, and other abnormal conditions. Any alarms would simply be announced through local speech synthesis, as there was no RF link, or even an associated operator control unit (i.e., remote host computer).

Not all situational awareness is derived from external sensors, however. An intelligent mobile platform intended to interact with humans needs an accurate time reference to assist in the execution of its behavior routines. Consideration must also be given to which internal checkpoints must be monitored, such as voltage levels, temperature, current flow, etc. Self diagnostics play an important role in the successful introduction of any new technology, not only during development, but perhaps even more so later during operation and maintenance. For this reason, I made it a point to try and instrument anything and everything that could potentially go wrong, using speech synthesis to announce the nature of the fault. Obviously it's best to address this feature up front during the early planning stages, rather than try to retrofit after the fact.

As the design begins to converge, the need for automatic replenishment of onboard energy becomes more readily apparent. Since battery power is generally viewed as the most practical supply for indoor operation, this usually suggests some type of homing system to locate a recharging station, and a reliable method of making the electrical connection. Judicious energy conservation becomes critical in order to achieve a practical run-time duty cycle. Circuits that are powered down when no longer needed, however, can often affect other still-energized systems, and any interconnection must take this into account.

1.2 Preliminary Design

Accordingly, the initial design of *ROBART I* provided for the following:

- Chassis and body framework with a tricycle wheelbase.
- A steerable (±80 degrees either side of centerline) front drive wheel.
- A rotating (±100 degrees) head assembly mounted on the body trunk.
- A scanning photocell array to track a homing beacon on the recharging station.
- A sonar for measuring range to any objects in the forward path.
- Multiple near-infrared proximity sensors for obstacle detection.
- Contact bumpers and feelers for collision detection.
- Numerous sensors for intrusion detection and home surveillance.
- Speech synthesis for communication with humans.

Although rich in collision-avoidance/detection sensors, this indoor research platform would have no sense of its absolute location, and was therefore limited to navigating along routes loosely defined by the relative locations of individual rooms with respect to a hallway.

1.2.1 Mechanical Structure

Since *ROBART I* was intended strictly as a learning tool, no attempt was made to provide an outer protective skin. All circuits and mechanical parts were to remain as open and accessible as possible without degradation of performance or undue risk of physical damage. The overall height of 57 inches was chosen to allow line-of-sight view of the recharging beacon over nominal obstructions, yet not so high as to preclude fabrication of two uprights from a 6-foot length of aluminum-angle stock (see Figure 1-4).

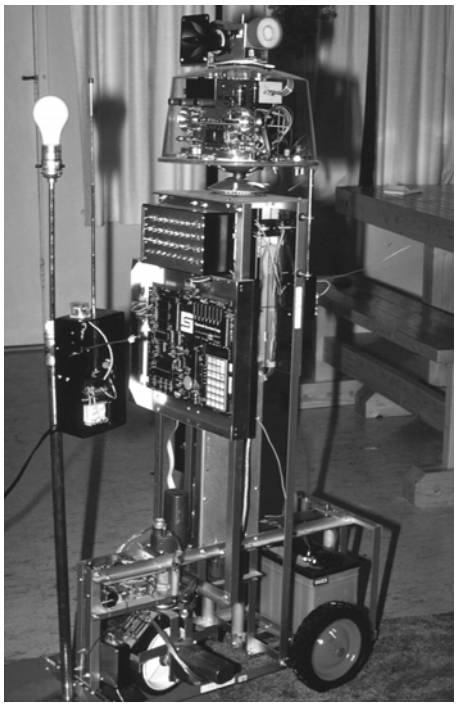


Figure 1-4. The overall height of *ROBART I* was chosen to facilitate an unobstructed view of the recharging beacon over typical household items such as tables and chairs.

The rectangular base measured 26 inches from front to rear bumpers. The width was set at 15 inches to allow sufficient room for the drive motors attached to either side of the front wheel, and to provide a stable foundation for the body trunk. The two rear wheels were recessed into the side of the base to provide a relatively smooth side panel free of any significant protrusions. The resulting narrow profile facilitated safe passage between obstacles and through doors.

A tricycle wheel-base was chosen, primarily for comparison with the track-drive mobility configuration used previously on the *CRAWLER* series. The maximum turn angle for the drive wheel was 80 degrees in either direction. This steering geometry yielded a minimum turn radius of 12 inches, resulting in a required free-radius of 21 inches in which to perform a turn without impact. While clearly not as maneuverable as a differentially steered system that could pivot in place, this arrangement did allow for *ROBART I* to execute a 180-degree Y-turn in a standard 36-inch hallway.

1.2.2 Onboard Microprocessors

Only two microprocessors were employed in the system, one to provide supervisory control and another dedicated to speech synthesis. The first was a *SYM-1* single-board

computer designed by Ray Holt of Synertek Systems, Inc., Santa Clara, CA (Holt, 2001). The *SYM* was chosen as the primary controller due to the extensive input/output (I/O) hardware available, an onboard *Assembler/Editor* for software development, and a relatively low total package price of just over $100.

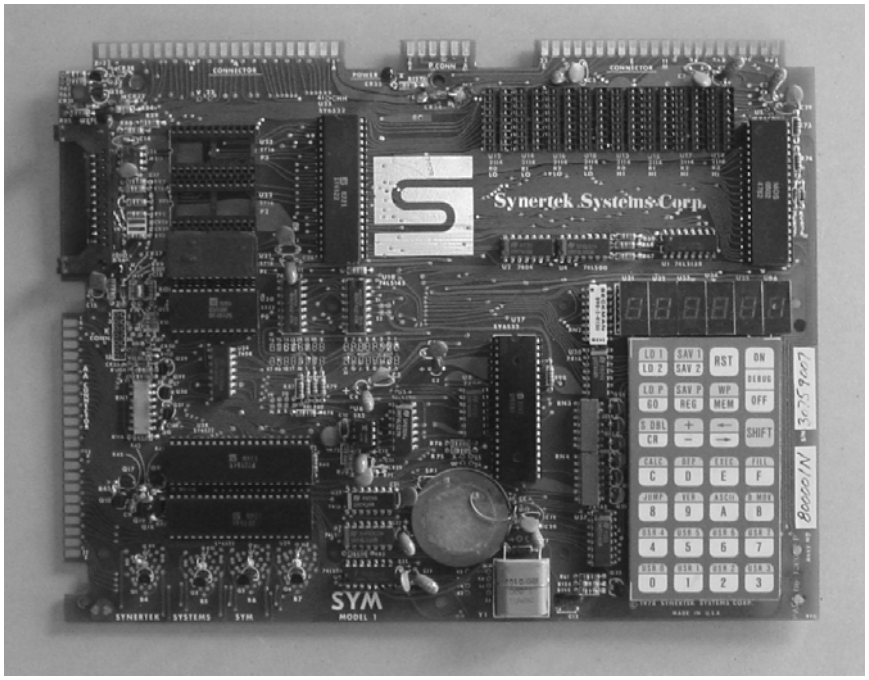


Figure 1-5. The Synertek *SYM-1* single-board computer, introduced as a development kit for the 8-bit 6502 microprocessor, was rich in I/O capability and therefore ideal for robotic experimentation.

The *SYM* was mounted at mid-height on the front of the robot, but could be interfaced to a Synertek *KTM-2* terminal through an RS-232 connection, thus greatly expanding the practicality of the overall setup. The *KTM* (for keyboard terminal monitor) provided a serial keyboard interface and composite-video output to a monochrome monitor, which was the de facto standard of the time (Figure 1-6). I actually wrote my thesis in this fashion, using a primitive word processor based upon the *SYM's* assembly-language editor, much to the envy of my fellow grad students. The robot remained motionless beside the terminal stand while the *SYM* was under *KTM* control, and in exchange received supplemental power over the same DB-25 connector that provided the RS-232 link. Once released, the software checked this connector before driving away, and operator assistance would be requested through speech synthesis if the serial cable had not been unplugged.

Figure 1-6. The Synertek *KTM-2* terminal and a cassette tape storage system are shown beside *ROBART I*, enabling the onboard *SYM* to be used for both software development and word processing (circa 1981).

The 6502 microprocessor on the *SYM* employed a 16-bit address bus and an 8-bit data bus, with 4 kilobytes of random-access memory (RAM) and 28 kilobytes of read-only memory (ROM) onboard. Three 6522 Versatile Interface Adapters (VIAs) and one 6532 Peripheral Interface Adapter (PIA) provided 71 I/O lines, making the *SYM* ideal for robotic applications. Off-board expansion was provided through a 44-pin expansion connector, and a Beta Computer Devices RAM-expansion board provided 32 kilobytes of additional memory. I later added an 8-kilobyte EPROM board of my own design to support a library of software routines to facilitate higher level coding.

1.2.3 I/O Expansion

The *SYM-1* communicated with the outside world through its three 6522 VIAs and the 6532 PIA, each of which contained two parallel eight-bit I/O ports (Port A and Port B), which could be used to read sensory information or to send logic commands to external circuitry (Figure 1-7). As the initial design began to materialize, it became apparent that even 71 binary I/O lines were insufficient, due to the vast number of inputs and outputs required for autonomous control of a complex mobile system. For example, 13 I/O lines were needed for communication with the speech-synthesis processor, 8 for head and drive-wheel position control, five for an onboard operator interface, and 16 for the tactile sensors.

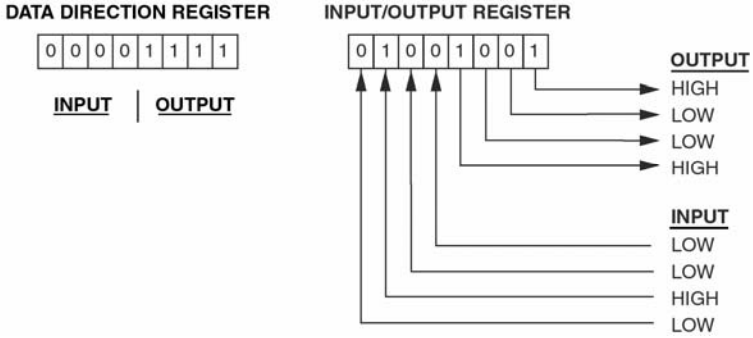


Figure 1-7. I/O line status as reflected by individual bit values in the *6522's* Input/Output Register.

Therefore, a four-line address bus was created to serve the six interface boards that connected the *SYM* to the various sensors and output devices under its control. This address bus drove three 74150 sixteen-input data selectors, two 74154 16-output data distributors, and one 7475 four-bit latch as shown in Figure 1-8. This arrangement yielded 84 more I/O lines, but at the expense of 10 of the original 71. Additional groups of 16 lines each could have been added at a cost of one I/O line per group, but this further expansion was never needed.

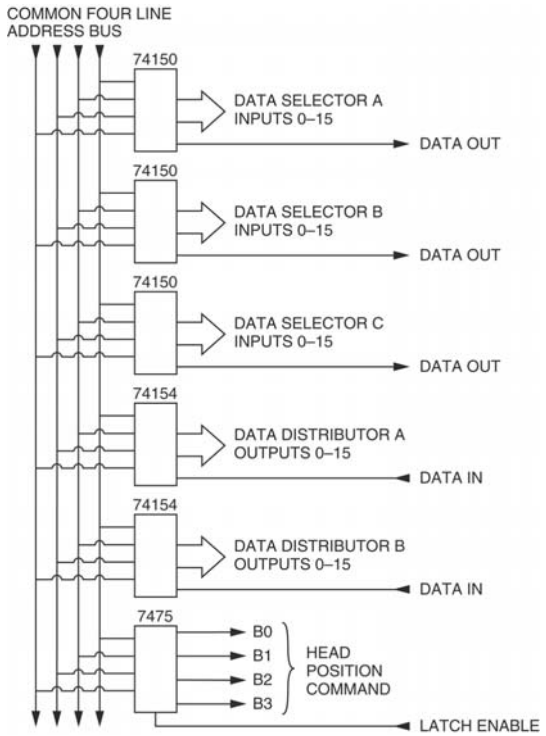


Figure 1-8. The four-line I/O expansion bus serviced three data selectors, two data distributors, and the head-position latch.

The three data selectors were simultaneously driven by the four-line address, with the complement of the selected input appearing on the output, pin 10 (Figure 1-9). The *SYM* would set the value of the desired input number on the four-line address, and then read the appropriate selector output over the corresponding 6522 VIA input. Subroutines *ReadA*, *ReadB*, and *ReadC* took care of manipulating the address and data lines for ease in programming, with the desired input number loaded into the X register before the appropriate subroutine was called. Data Selector A was entirely dedicated to tactile and proximity sensors, and was interrupt-capable. Data Selector B, also interrupt-capable, handled all alarms and internal-circuitry check points. Data Selector C was used to poll miscellaneous inputs and had no interrupt capability.

The two sixteen-output 75154 data distributors were also driven by the four-line address bus, but required three additional control lines, referred to as *Data 1*, *Data 2*, and *Data 3*. These three lines were normally maintained in the high state. *Data 1* and *Data 2* provided the inputs to Distributor A and Distributor B, respectively, and could be sent low to pull down the desired distributor output. All 16 outputs were normally high, except the selected output, which followed the applied input. Thus, if the inputs were kept high, no outputs would change state, regardless of the status of the four-line address.

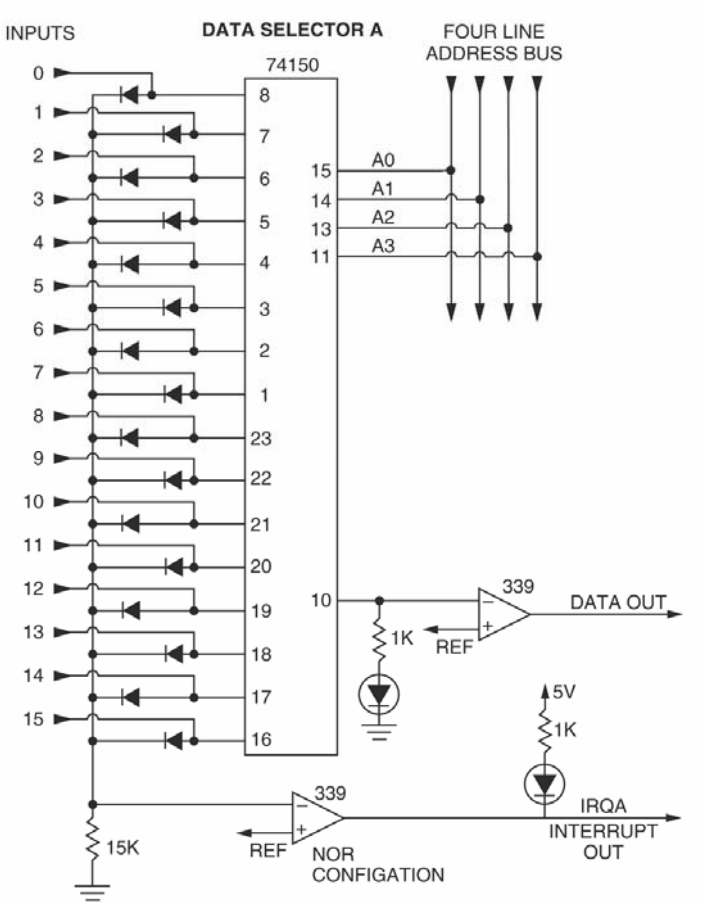


Figure 1-9. Schematic diagram of a multiplexed 16-input Data Selector used to increase the number of binary I/O lines available for input.

If the *SYM* desired to momentarily pull low an output on Distributor B, it first set the appropriate binary value on the four-line address bus, and then pulsed *Data 2* low. Output 3 on Distributor B would also go low, but output 3 on Distributor A would be unaffected, since it followed *Data 1*. This system could only be used to strobe outputs on Selector B, in that they could not be held low for any length of time without tying up the CPU. Distributor B outputs therefore were used only as active-low triggers to initiate timing sequences or actions subsequently controlled by other circuitry.

The third control line was used with Distributor A to overcome this problem and provide a means for latching the output high or low. As shown in Figure 1-10, each output on Distributor A was used to clock a D-type flip-flop. These flip-flops all had their D inputs connected to *Data 3*, so their outputs would assume the logic level of *Data 3* if and only if that flip-flop was clocked by Distributor A. The clocking sequence involved selecting the desired flip-flop, with Data Lines 1-3 high. *Data 3* was then set to the desired output logic level (high or low). *Data 1* was then strobed to clock the flip-flop, which assumed and held the desired state dictated by *Data 3*. This rather complicated manipulation of address and control lines was all done by subroutines *Pin.hi* and *Pin.lo*.

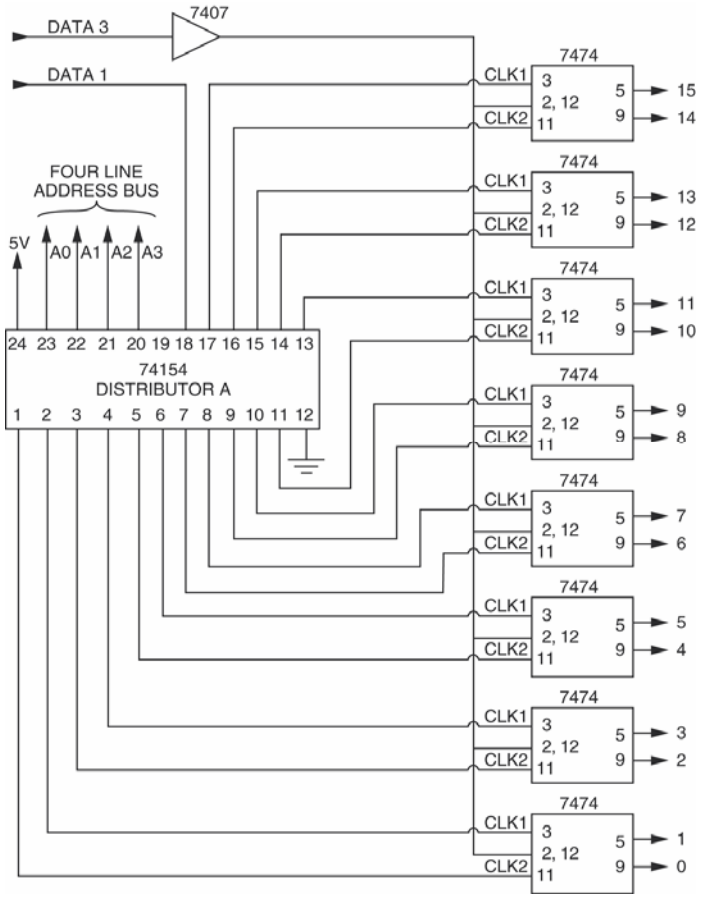


Figure 1-10. Schematic diagram of a multiplexed 16-output Data Distributor used to increase the number of binary output lines.

1.2.4 Drive and Steering Control

ROBART I was propelled by two surplus gear motors (originally designed for use as valve actuators), each with a separate forward and reverse winding. The motors were energized by an electro-mechanical relay controlled by PA4 of 6522-2, and their direction was similarly controlled by PA5 of the same port. At 12 volts DC, output shaft rotation was 10 rpm under full load, yielding a very conservative forward velocity of approximately 16 feet per minute, so no attempt was made to provide any speed control. These motors were selected for their extremely low cost, high torque, and ready availability as surplus items, as opposed to optimal performance characteristics. They were mounted on either side of what was referred to as the drive-wheel support cage, as shown in Figure 1-11. This cage, in turn, was supported by a vertical steering column and thrust bearing, and could be rotated by a steering motor (identical to the motors used for propulsion) up to 80 degrees on either side of centerline. The entire drive assembly thus pivoted around the steering column with its angular position sensed by a belt-driven potentiometer.

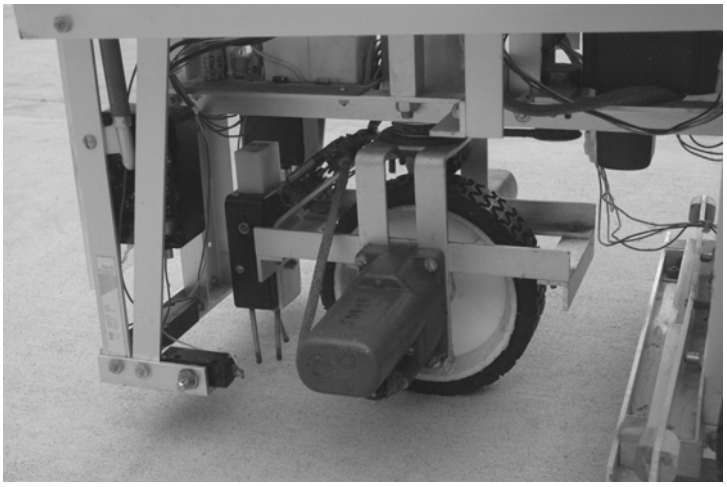


Figure 1-11. Tandem drive motors were attached to each side of the front wheel for forward and reverse motion, while a third motor situated directly above provided ±80 degrees of steering angle.

Due to monetary constraints, an extremely simple position-control system was chosen that required only four bits of resolution, effectively creating 16 discrete steering angles approximately 10 degrees apart, as previously done by Dacosta (1979). The four output bits from the analog-to-digital (A/D) converter were fed to the B inputs of a 7485 four-bit magnitude comparator, as shown in Figure 1-12. The desired steering position was fed to the A inputs by 6522-2 (PAO – PA3). The comparator compared the two numbers, and the motor was energized as long as the actual position was not equal to the desired position. The direction of rotation was determined by whichever value was greater.

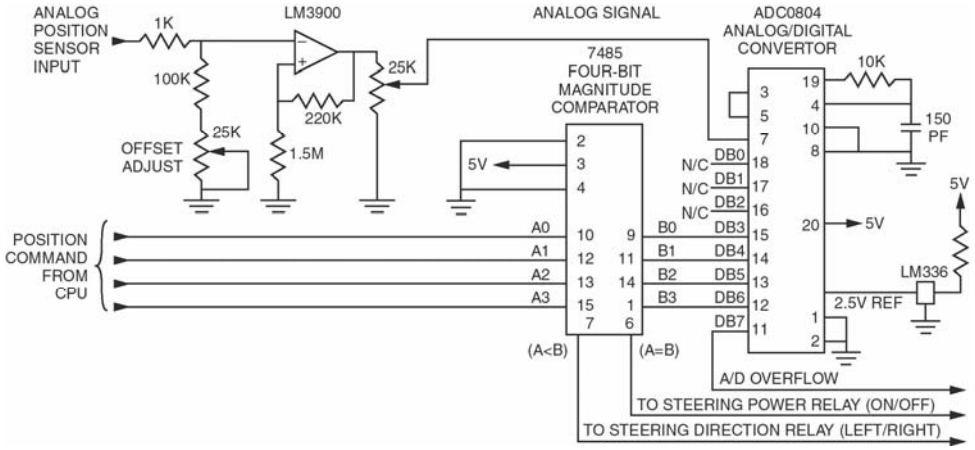


Figure 1-12. A hardware-based digital servo-controller was used for steering-angle positioning.

The sensing potentiometer used for steering-angle feedback was wired as a voltage divider across a regulated 5-volt supply, and configured to provide a linear output ranging from 0 to 3.7 volts as the support cage turned from right to left. This voltage was fed to an operational amplifier for isolation purposes, and the amplifier output applied across a 25K potentiometer used to set the maximum output to 2.5 volts (wheel full left). The output from this second potentiometer was digitized by a National Semiconductor® *ADCO804* A/D converter. The *ADCO804* was designed for an input swing of 0 to 5

volts, so the most significant bit was not used. This configuration effectively created an A/D converter with a range of 0 to 2.5 volts, but with only seven bits of resolution.

As previously pointed out, this was a very crude positioning scheme, with rather coarse resolution and no velocity control, but it proved more than adequate for use in the homing and navigational routines to be presented later. A considerable amount of frictional damping was inherently present, which decreased the chances of overshoot, especially since there were only 16 discrete positions. Had more steering resolution and/or faster response been desired, however, overshoot would have become a significant problem, hence requiring some form of velocity control.

1.2.5 Head-Position Control

The final device serviced by the four-line address bus was a level-sensitive quad latch used to store a four-bit command for head positioning, controlled by PB7 of 6522-2 (referred to as *Latch Enable*). When PB7 was high, the latch contents would reflect the value of the four-line address bus, and subsequently held that value when *Latch Enable* went low. Thus, any four-bit number could be latched into this register as a head-position command, and dedicated logic circuitry (identical to that of the steering-motor presented in Figure 1-12) would cause the head motor to pan accordingly without the *SYM* being further involved. Additional modes were provided to facilitate detection and tracking of the recharging station beacon (see section 3), and scanning ahead for open space in support of navigational planning (see section 5).

1.2.6 Speech Synthesizer

ROBART's computer-generated voice output was implemented using National Semiconductor's *Digitalker* speech-synthesis system, based on their *MM54104* speech processor chip with two sets of vocabulary instructions (280 words) stored on EPROMs (Smith & Weinrich, 1980). A fixed vocabulary was preferred over unlimited text-to-speech created through repeated use of phonemes, due to the decreased demand on the host microprocessor (in terms of execution time and memory space). The *SYM* controlled the *Digitalker* over two parallel I/O ports, one of which supplied the EPROM index for the desired word. A portion of the second I/O port was used to initiate the speech output and to detect completion of each spoken word.

Subroutines *Vox1* and *Vox2* requested words from vocabulary lists 1 and 2, respectively, just as *Vox3* was to deal with those words on list 3 when its associated EPROMs became available. (By the time these EPROMS were introduced, I had already started work on *ROBART II*, so *ROBART I* was limited to the original 280 words.) The hex address of the desired word was first loaded into the *X* register of the 6502, and then the appropriate subroutine called (*Vox1* or *Vox2*). Subroutine *Talk* was invoked by the chosen *Vox* routine to manipulate the *Digitalker* control lines to initiate speech, and then waited for the busy signal to clear. Alternatively, subroutines *Vox1d* and *Vox2d* could be called if a slight delay was desired before outputting the specified word, for spacing between consecutive words in a sentence.

Note that there were not a lot of computer-generated speech applications in 1980 when the *Digitalker* was first introduced. I, of course, ordered the kit version to save money, and had it fully assembled on my bench in just a few short hours. I hooked up a 5-volt power supply, tied all the input pins low to select word 00, and then strobed the "write"

line to see if it worked as advertised. I vividly remember the eerie feeling as that tinny little voice sprang to life and proudly announced: "This is *Digitalker*." I must have spent the next 2 to 3 hours changing the eight flea-clip jumpers to activate each and every word on both lists, some more than once. By midnight the *Digitalker* was successfully interfaced to the *SYM*, whereupon *ROBART I* could actually talk under computer control! Seems pretty commonplace by today's standards, but I was too excited that night to ever fall asleep.

1.2.7 Real-Time Clock/Calendar

Another computer feature we now take for granted is the real-time clock/calendar used by the operating system, but back in the early eighties, you had to roll your own. *ROBART I's* clock was implemented in software, but derived its timing pulses from hardware circuitry located on the *Clock-Calendar* board in the head. A National Semiconductor® *Clock/Temperature Module* was externally driven by a quartz oscillator (via a 17-stage divider for a precise 60-Hertz reference), and would show the time in hours and minutes on its seven-segment display (Figure 1-13). The temperature in either Celsius or Fahrenheit could also be displayed.

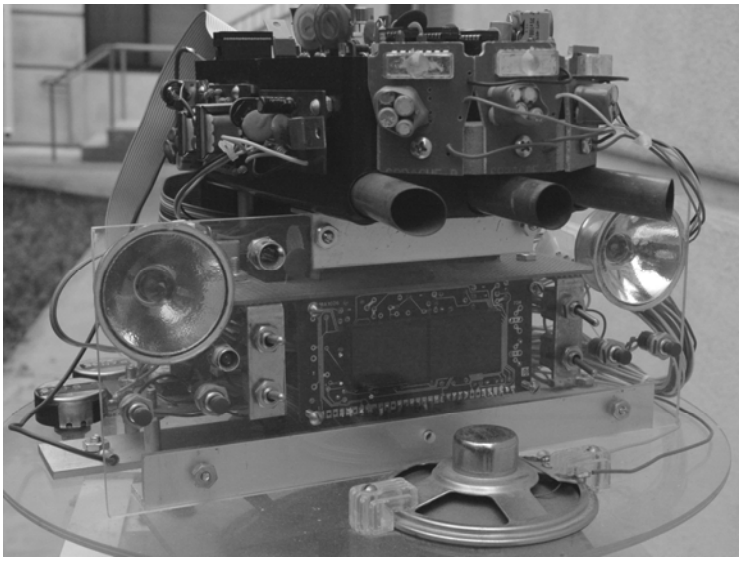


Figure 1-13. A National Semiconductor® *Clock/Temperature Module* mounted in the head (shown cover removed) provided a 60-Hertz reference for the real-time-clock software running on the *SYM*.

The *Clock/Temperature* module provided only a one-bit digital output, AM/PM, intended to drive an external LED. I used it instead to clock a binary counter, which incremented from 0 to 6 and then reset. This count represented the day of the week (Sunday = 0), and drove an additional seven-segment display located on the *Clock/Calendar* board. The day-of-week count was also read by Data Selector C for use by the software. The *SYM* had no direct way of reading the hardware *Clock/Temperature* module to ascertain the hours, minutes, and seconds, however.

To get around this problem, the same 60-Hertz reference was fed to a divide-by-60 counter to produce a 1-Hertz square wave, which in turn was applied to the non-maskable interrupt (NMI) input of the 6502, thus generating an interrupt once each second. The NMI interrupt service routine incremented the software registers for *Seconds*, *Minutes*,

and *Hours* to duplicate the internal registers of the clock module. If initialized to the time shown on the *Clock/Temperature* display at system start up, the NMI routine would ensure that its registers tracked this time for subsequent use by the *SYM*.

Whenever the *Minutes* register was incremented, the NMI routine also cleared register *IRQ.fre*, which was used to count the number of maskable interrupts per minute. (This register provided a very useful piece of information for the collision avoidance routines, as will be discussed in the next section.) If incrementing *Minute* produced either a "00" or "30," the appropriate speech-synthesis word index for that value would be assigned to register *Timeflag*, for reasons discussed below. The NMI routine took no other action, thus ensuring the time required to service an NMI interrupt was kept as short as possible.

Subroutine *Time* was a speech-synthesis program intended to announce the time on the hour and half-hour, but without interruption of any critical routines that may have been in progress. This behavior coordination was handled by subroutine *Clock*, which could be called at appropriate points in the main program flow when voice output would not interfere with other activities. *Clock* first checked *Timeflag*, and returned with no action taken if the register had not been set by the NMI real-time clock software as discussed above. On the other hand, if the *Timeflag* register was set, voice output took the form: "The correct time is ___ hours and ___ minutes."

Needless to say, this clever feature got real old real fast, particularly after the household had retired for the evening, and so a second conditional was added to cancel voice output during the dark of night. Subroutine *Clock* would first check the ambient-light photocell on top of the head, and if the room were dark, *Timeflag* would be cleared and no announcement took place. This low-level behavior-suppression capability almost caused a real problem during my thesis presentation some 18 months later, however. I had incorporated quite a bit of computer-generated speech into a live-onstage demonstration for the entire Mechanical Engineering Department, figuring the more the robot said, the less I had to do. But when the lights were dimmed for my accompanying slide show, *ROBART* assumed the assembled audience had all gone to sleep, and respectfully refused to talk! Fortunately I figured it out soon enough, and recovered smoothly, but the near-miss made a lasting impression that was to significantly influence my future designs with regard to more robust self-diagnostics.

1.2.8 Software Considerations

Up to this point I had been implementing software in more or less a stand-alone fashion, building up a basic library of device drivers and action primitives that could collectively support a variety of more sophisticated behavior routines still to come. In the course of this effort, it became readily apparent that competing tasks must somehow be arbitrated to allow emergent needs of greater urgency to override less important activities already in progress. The most obvious example here would be interruption of a goal-driven navigational behavior in order to avoid an obstacle. Once the obstruction was clear, the original objective must be recoverable and then resumed. Making such behavior arbitration happen in a smooth and orderly fashion with extremely limited resources will be the focus of our next section.

1.3 References

Brooks, Rodney A., *Flesh and Machines: How Robots Will Change Us*, ISBN 0-375-42079-7, Pantheon Books, Random House, New York, NY, 2002.

CMU, http://www.robothalloffame.org/04inductees/shakey.html, 2004.

Dacosta, F., *How to Build your own Working Robot Pet*, Tab Books, pp. 86-87, 1979.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot*, Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995.

Frederiksen, T.M., and Howard, W.M., "A Single-Chip Monolithic Sonar System," *IEEE Journal of Solid State Circuits*, Vol. SC-9, No. 6, pp. 394-402, December, 1974.

Holt, Ray, www.microcomputer history.com, e-mail correspondence dated February 5, 2001.

Moravec, Hans P., "The Stanford Cart and the CMU Rover," *Proceedings of the IEEE*, Vol. 71, pp. 872-884, 1983.

Nilson, J.J., *Shakey the Robot*, Technical Note 323, SRI AI Center, Menlo Park, CA, 1984.

Smith, Jim, and Weinrich, Dave, "Speech Synthesis," Application Note 252, National Semiconductor, Santa Clara, CA, December, 1980.

2
Collision Avoidance

"Man is the most versatile servo-mechanism
mass produced by unskilled labor."

Author unknown

From an evolutionary perspective, once an autonomous robot becomes capable of motion, close attention must shift quickly to the issue of collision avoidance. At this critical point in the development of my "thesis creature," I envisioned some high-level deliberative routine that drove the robot towards a purposeful goal, such as connecting with its recharging station. But concurrent with that I wanted some protective over-watch mechanism, always running in the background, to keep the moving platform from hitting anything. I did not want to be embedding repetitive collision-avoidance conditionals in every new behavior routine ultimately to be generated by my slowly improving programming skills. There had to be a better way.

2.1 Subconscious Over-Watch

The scheme I envisioned was patterned directly after Nature, very much along the lines of the *conscious* and *subconscious behavior* we exhibit as human beings. A good analogy is seen in humans driving their automobiles, if you consider the driver as playing the role of the robot's computer and its associated sensor subsystems, and think of the car as representing the mobility base. How many times have we been totally lost in thought, perhaps pondering some bothersome issue from work, suddenly to realize that we just missed our exit off the freeway? [2] Who (or what) was driving the car while we were daydreaming? The answer is our *subconscious mind*. A kid darts in front of us as our *conscious mind* plots the business strategy we will present at the upcoming meeting, for which we are running late and hence maybe driving a bit too fast. Instantly our overriding *subconscious* reacts, interrupting our daydreaming and refocusing all available resources on avoiding a terrible tragedy.

[2] During extensive literature searches in preparation for the final few chapters of *The Evolution of ROBART*, I subsequently learned this same example had been formally treated by Armstrong (1981), and later expanded by others (Lycan & Ryder, 2003).

The exact same thing would hopefully happen if we were fumbling with the rear-view mirror (instead of thinking about work), and the same kid makes his move. Or perhaps we are busy dialing a cellphone with our visual acuity concentrated upon the keypad, leaving only peripheral situational awareness of events outside the car. The point is, we could be distracted by any number of alternative behaviors, and the same *subconscious* routine still keeps us safe. We do not have to think about it each time in the unique context of a new activity, which is a good thing, because sooner or later we would forget to do it. Such a *subconscious "guarded-motion" strategy* evolved in humans (and before that, animals) over billions of years for but one reason: it very effectively promoted survival. Hence it sounded to me at the time like a good model upon which to build a collision-avoidance strategy for *ROBART I*.

2.1.1 Subconscious Perception

The problem basically boils down to information overload. Conscious awareness is by nature a serial process, and there are just too many things to perceive and assess in order to survive in such fashion. Herman Spitz (1997) explains it as follows:

> "Because it is impossible to attend to two events at the same time (although we can alternate rapidly between them), it is essential that motor movements be executed involuntarily while we attend to impinging events. No organism can survive without muscle movement that bypasses awareness and conscious control. For the simplest organisms, that is all there is. In evolutionary development, therefore, nonconscious movement must have preceded the development of consciousness."

A rather illustrative example of *subconscious perception* was unknowingly presented to me the other day when a young friend asked if I had heard about "that 1-1-1 thing," a supposedly paranormal experience involving the viewing of digital clocks. Every time she looked at a clock, it seems, the display read "1:11." We look at the clock on a continuous basis, I explained carefully, we just don't usually care that much what time it really is. In fact, we *subconsciously* look at everything all around us, but *consciously* we ignore those things for which we have had no prior reason to be especially interested. Dretske (2005) labels this phenomenon as "perception without awareness." But if someone has specifically called our attention to some bizarre "1-1-1" phenomenon, then you can be sure that the very next time our *subconscious mind* sees such displayed, it will dutifully direct our *conscious* gaze right to the offending clock. And voila, there it is, precisely as predicted!

In all fairness, *subconscious perception* is not that well understood, even by the more properly educated. In thinking about it later, I realized I had probably experienced a similar "1-1-1" phenomenon of my own, had I only but known. It seems like every time I look at the gas gage in my car, it says "E!" So many times I just "happened" to notice this situation, usually not a moment too soon, in that I was basically running on fumes. I remember thinking to myself on more than one occasion how lucky it was that by chance I had perceived this dilemma in the nick of time, thus narrowly averting an embarrassing, inconvenient, or perhaps even dangerous mishap. But was it in fact chance? Or had my subconscious perception been monitoring the gage all along, and only brought such to my

attention when it began to matter? If this were indeed the case, it would sure be nice to get a bit more advance notice in the future!

Here is another example, albeit from a slightly different take. Imagine sitting in close proximity to some object, so close you can easily reach forward and touch it, even hold it in your hands. In fact, you often do hold it, sometimes for hours at a time, and this has been happening for years on an almost daily basis. Yet oddly enough, you have very little idea what this mysterious object looks like, even after being told what it is. In the countless robotic presentations I have given over the years, I have on several occasions asked members of my audience to draw a simple diagram of the steering wheel of their car. Once the basic outer circle has been rendered, most people are unable to add a whole lot more, and some cannot even recall the color!

We sit for hours in rush-hour traffic, holding the steering wheel firmly in our hands, yet it seems we never actually look at it. Our conscious attention is forever directed elsewhere, usually outside the vehicle. We do not look at the steering wheel because we generally have no real reason to do so. (Although more recently I have noticed this has started to change with the advent of embedded climate, entertainment, and cruise controls.) In other words, we can only consciously process so much, and if that was all we could do, it would not be nearly enough. As a consequence, most of our perception of the surrounding environment is done subconsciously, and all those details that would otherwise distract from the real issues are filtered out, so that our limited focus of attention can be more effectively engaged. Spitz (1997) again sums it up nicely:

> "Not only are conscious mental operations limited, but attention is also limited. These are major bottlenecks. One way in which these bottlenecks are eased is the spontaneous assignment of well-learned activities to another system, one that does not use precious space in consciousness. Another is to supplement learning by gathering information outside of conscious attention."

Such a perceptual-analysis limitation was precisely the problem I was facing with the primitive computational resources available in the early eighties, particularly on a self-financed thesis project. Hence, I decided to emulate the biological approach with regard to the collision avoidance strategy on *ROBART I*, in the form of *subconsciously executed* reactive responses governed by *subconsciously gathered* proximity and tactile data. No doubt I was also inspired toward this end in reading the multiple reference manuals that came with the *SYM-1* single-board computer, learning in the process about software interrupts (Synertek, 1976).

For those not familiar, an interrupt is an event that is hardwired to force the processor to drop what it was doing and service the cause of the interrupt. By analogy, *conscious* attention is redirected by the *subconscious*. No polling of devices is required until an interrupt is detected. The processor is alerted to this condition by special input lines connected to the hardware that it services, *6522 Versatile Interface Adapters* in the case of the *SYM* (i.e., IRQA and IRQB in Figure 2-1). This strategy leads to much more efficient operation, as the processor need not concern itself with any interrupt sources until such time as they actually require attention. A good analogy here would be

answering the phone only when it rings versus lifting the handset on a regular basis all day long just to see if someone is calling.

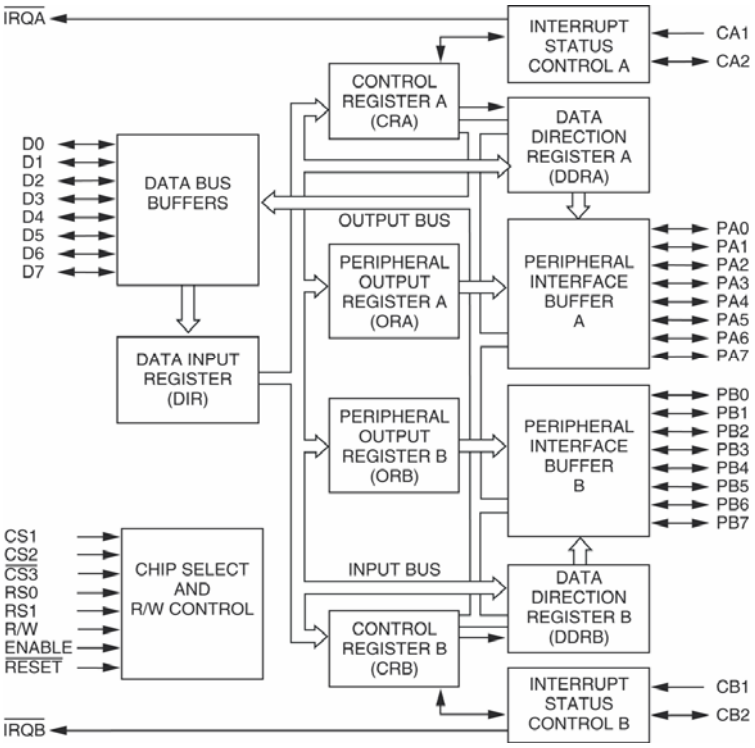


Figure 2-1. Block diagram of the *6522 Versatile Interface Adaptor* used on the Synertek *SYM-1*.

2.1.2 Robotic Perception

I am a firm believer in the bottom-up school of design, where you start with the fundamentals, and then incrementally improve functionality with an evolutionary upgrade approach. So I basically wanted to pick up where I had left off in 1966, now that the supporting technology had matured to the point where I could buy a single-board computer for just over $100. The first deliberative behavior implemented on *ROBART I* was simply patterned after the wandering nature of *CRAWLER I*: moving mindlessly forward and changing course only as necessary to avoid collisions. I specifically did not mention this ancestral connection in the thesis write up, so as not to suggest my graduate research was merely an extension of my high-school science project.

But now that the statute of limitations has more than run its course, I freely admit the initial objective was essentially just that, upgrading the *Wander* routine of *CRAWLER I* with better reactive instincts. To me there were two complementary issues: (1) improved sensors for better situational awareness (versus just four crude tactile feelers), and (2) more sophisticated assessment of this sensor data, using an actual computer for planning the response (versus an overly simplistic punched-card reader). I already had the computer, although it was not yet programmed to do anything. Before writing any software, I had to provide an appropriate suite of sensors. I opted for a multi-level scheme of range, proximity, and impact detection, with numerous sensors installed at

appropriate points on the chassis structure to optimize the surrounding envelope of protection (Figure 2-2).

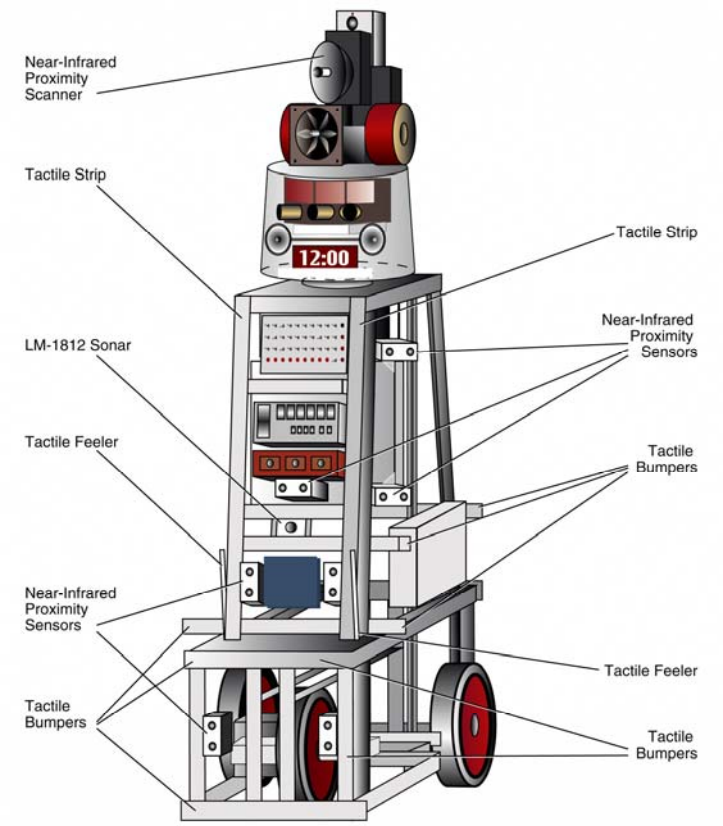


Figure 2-2. *ROBART I* was generously equipped with a variety of proximity sensors, feeler probes, and tactile bumpers for collision detection and avoidance.

Long-range information (i.e., 1 to 8 feet) was derived from forward-looking sonar. A near-infrared proximity scanner mounted on the head reliably sensed diffuse surfaces out to about 5 feet, with fairly good angular resolution but no range information. A 10-channel proximity-detection system provided close-in protection (i.e., to about 18 inches). Tactile feelers at critical points around the base periphery sensed impending collisions, while contact bumpers incorporated into the base and body trunk alerted the software to any actual impact. As a final backup, the drive motor was monitored for an over-current condition indicative of a stall. Brief descriptions of these sensors are provided below, presented in the order of actual implementation.

Sonar Sensors

The first collision-avoidance system actually installed on *ROBART I* was a 21-kHz rangefinder built around the National Semiconductor® *LM1812* monolithic sonar transceiver, since the now-popular Polaroid ultrasonic ranging module had not yet been introduced. The 18-pin chip contained a pulse-modulated class-C transmitter, a high-gain receiver, a pulse-modulation detector, and noise-rejection circuitry, with an advertised maximum range of 100 feet in water and 20 feet in air (Everett, 1995). Circuit operation (see Figure 2-3) is described in detail by Fredericksen and Howard (1974). This sonar provided range information to objects directly in the robot's intended path of travel.

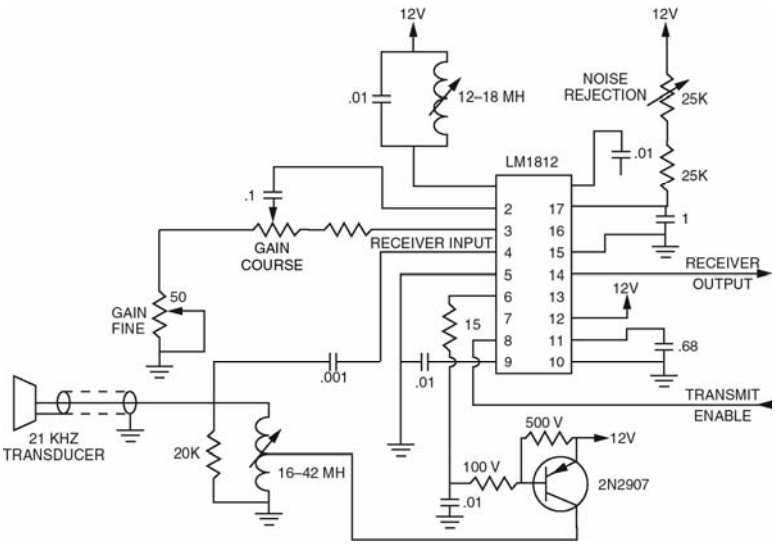


Figure 2-3. Originally intended for use in a fish-finder application, the National Semiconductor® *LM1812* sonar transceiver (now obsolete) is shown here configured for operation in air.

Early tests quickly showed the need for greater awareness of the immediate forward-path environment than provided by sonar data alone, since the in-air performance of the *LM1812* configuration shown in Figure 2-3 was somewhat limited. Although large items such as walls and furniture were generally detected, smaller objects often passed unnoticed below the beam pattern, or reflected too little energy for reliable detection. Additionally, no optimal decision could be made as to which way to turn, since the sonar transducer was mounted to the chassis in a fixed centerline orientation, with no capability to scan back and forth (Figure 2-4).

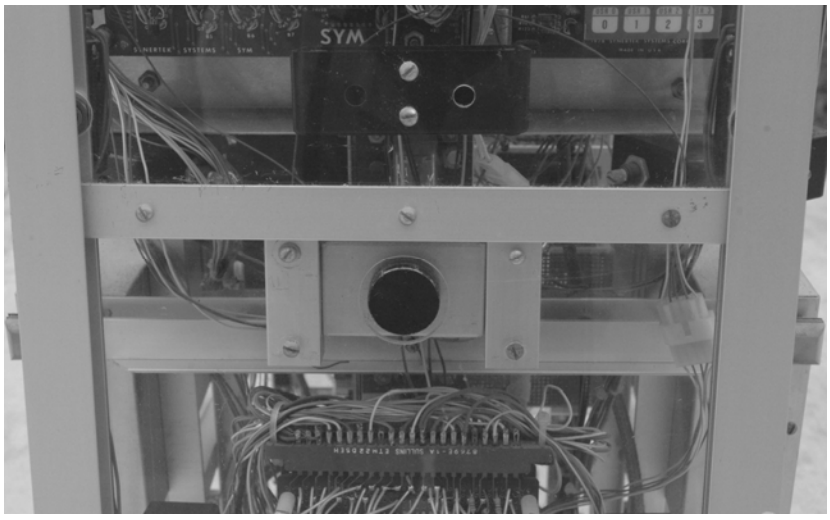


Figure 2-4. A Massa piezoelectric sonar transducer, mounted 20 inches above the floor on the front of the robot (photo center), provided range information out to about 8 feet.

Tactile Sensors

My first attempt to improve collision-avoidance involved revisiting the tactile sensors I had used on the *CRAWLER* robots, the intent being to improve the mechanical

implementation and significantly increase the number for better performance. I started by installing a dozen or so feeler probes around the perimeter of the robot base, each extending out 6 to 8 inches. These units were constructed from ordinary automobile curb feelers (not so ordinary any more), and were flexible so as not to cause damage to any surrounding objects. A cylindrical metal sleeve was fitted around the lower end of the feeler and electrically insulated from it by means of a short length of plastic tubing, as shown in Figure 2-5. Any significant deflection of the feeler probe caused it to come into contact with the upper lip of the coaxial metal sleeve, completing the circuit to generate an active-low output.

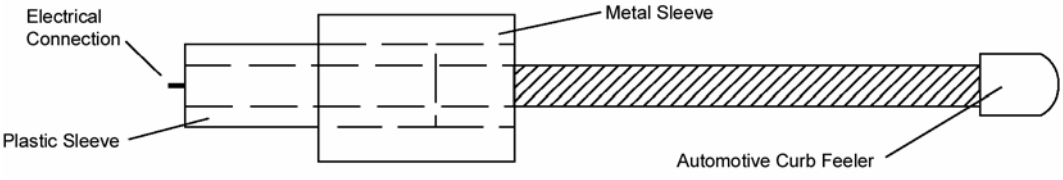


Figure 2-5. The tactile probes on *ROBART I* were fabricated from automobile curb feelers. Sensitivity was adjusted by sliding the metal sleeve towards (more sensitive) or away (less sensitive) from the tip.

The intent was to supplement the rather sparse data obtained from the sonar, and the probes did provide some improvement, but there was considerable room for more. (Problems arose, for example, from the inertia of the feelers themselves causing false activation of the detection circuitry when jarred during vehicle motion.) Furthermore, the task of protecting an almost 5-foot-tall robot with tactile probes was considerably more complicated than the case of the low-profile *CRAWLER* platform. As the number of feeler probes increased, vibration-induced false triggering rose to an unacceptable level, and *ROBART I* began to resemble a mechanical porcupine. In looking for a more aesthetic and robust solution that addressed these concerns, I designed a new front panel that was mechanically floated on a spring suspension and served as a surface-contact tactile sensor (Figure 2-6).

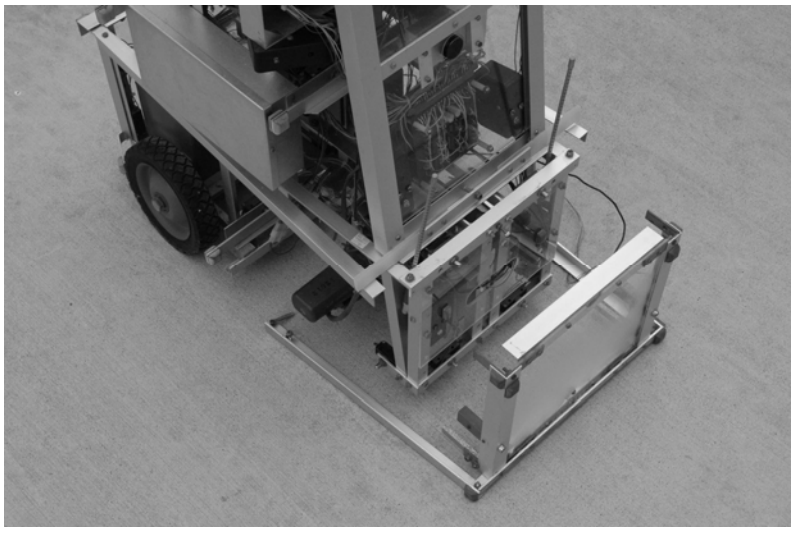


Figure 2-6. Two tactile feeler probes extend vertically from the front corners of the mobility base frame, just forward of the slanted upper-body structural supports on either side. The front panel assembly (shown detached) was mechanically floated to activate switch closures upon impact on the left, right, or front.

Flexible extensions protruding from either side of the base provided front-and-back coverage for the rear wheels (Figure 2-6). Since these spring-loaded probes could be activated by either forward or reverse motion, the collision-avoidance software first checked the direction of travel at time of impact, then responded accordingly. In addition, all leading structural edges were protected by hinged sections of aluminum angle that would actuate recessed micro-switches in the event of contact. The horizontal strips protecting the upper and lower edges of the mobility base from side impact during turns can be seen in the disassembled view of Figure 2-6, while the vertical strips protecting the structural uprights are portrayed in Figure 2-7 below. The result was much more subtle than the original porcupine configuration, less susceptible to false triggering, and more effective in terms of overall coverage.

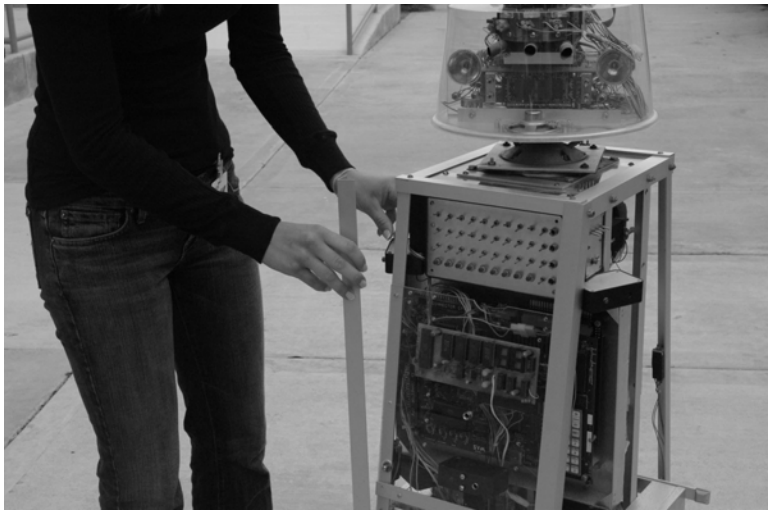


Figure 2-7. The four vertical structural members supporting the head assembly were protected by leading-edge tactile strips. The right-front strip (photo center) is shown temporarily detached from its upper restraint, revealing the micro-switch sensor underneath (photo center, bottom).

Proximity Sensors

A more elegant refinement, however, would be to sense and avoid a pending collision without actual feeler-probe contact, which had been the driving motivation behind the sonar. But the *LM1812* design I had been testing required a computer interface to calculate range from elapsed time, was somewhat problematic in its performance, and too expensive for full-volume coverage (i.e., using multiple units). I wanted something small, cheap, and reliable, and just to sense the nearness of an imposing obstacle, not actually calculate the range offset. In other words, I needed a *proximity sensor*, which sounds pretty obvious today, but in 1980 I was basically starting from scratch. In looking back on it, I am sure there were some commercially available units in existence at the time, but they would have been designed for industrial applications and therefore rather expensive.

So naturally I set out to build my own. I had already decided it would be much easier to track the optical recharging beacon with a three-element photocell array, as opposed to the single-cell detector employed on the *CRAWLER* robots. I chose to implement this approach in the form of three diverging collimating tubes situated on the head as shown

in Figure 2-8. (We will examine this beacon-tracking system in more detail in the next section.) Similarly, I had already installed a pair of 6-volt flashlight reflectors on the head faceplate on either side of the clock display (Figure 2-7), intended for general illumination in support of the security application (section 4). For theatrical reasons, I had also provided a circuit to flash these lights on and off, which my two young children found to be the most impressive feature of all on my computer-controlled robotic obsession. It later occurred to me that perhaps I could use the existing photocell array to detect returned energy from the flashing lights, and thus infer proximity in the direction the head was facing.

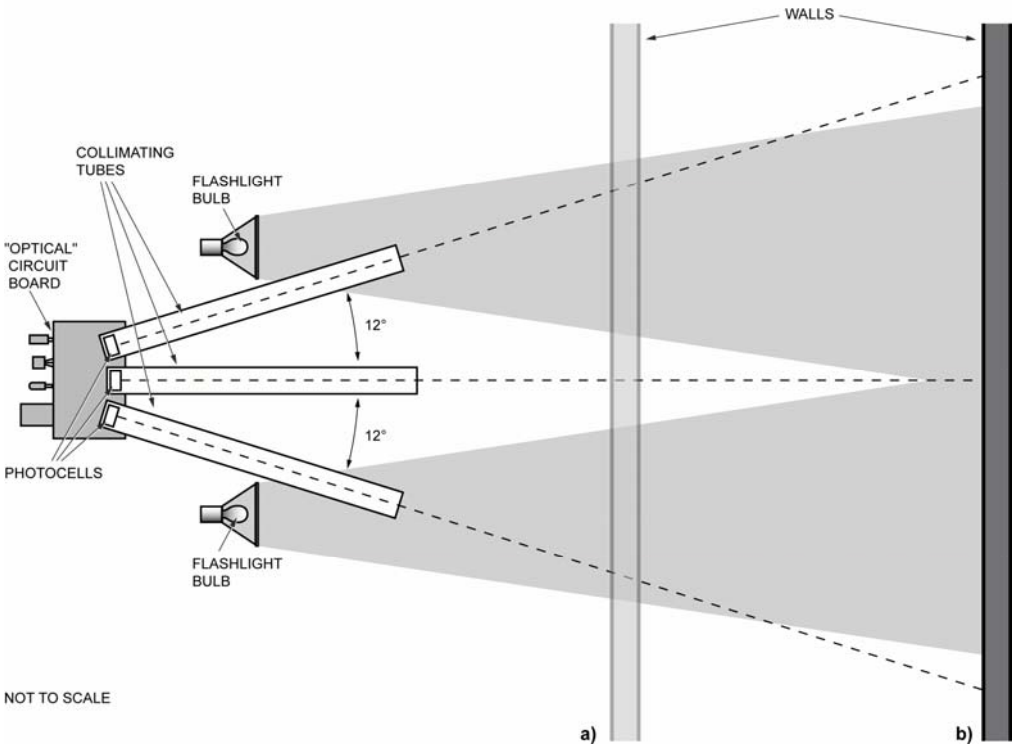


Figure 2-8. a) The outermost collimating tubes see more returned energy than the center tube for the near wall; **b)** For a wall approximately 3 feet away, the opposite is true.

Because of the serendipitous geometrical arrangement of these adjacent but unrelated systems, a very interesting effect was observed when I tested this theory, as illustrated above. If the robot was approximately 3 feet away from a perpendicular wall (Figure 2-8b), the only photocell to observe any significant increase in reflected energy was that associated with the center tube. This was because the optical axes of the two outer tubes diverged to either side of the main footprint of illumination. But as the robot moved closer to the wall (Figure 2-8a), these outer tubes eventually came into alignment with the areas illuminated by the two spotlights and their associated photocell outputs began to rise, while the center photocell output decreased. This situation reflects the basic triangulation principle behind the convergent proximity sensor (Everett, 1995), though I did not know this terminology at the time. Since the two outer photocells provided redundant information, I envisioned a simple "near-far-clear" implementation could be achieved with just two tubes, or better yet two small lenses. But the prospect of turning *ROBART I* into an annoying Christmas tree of flashing lights made the porcupine-like tactile probes seem somehow way less offensive.

As fate would have it, however, a much more elegant solution presented itself, again as an indirect result of an entirely unrelated intention. I had ordered a surplus electronic alarm enunciator to use as a warning siren for the security application, and this advertised feature turned out to be located on a multi-function circuit board from a discontinued toy spaceship (known as the *Star Bird*). There was no documentation aside from how to invoke the siren, but I noticed a clear LED emitter mounted next to a PIN photodiode, with what appeared to be obvious attempts at optical as well as electrical isolation. After playing around with it some I was able to get current flowing through the LED (as observed on my scope), but with no signs of visible illumination.

I eventually noticed that while the LED was firing, if you pointed one of these intriguing little spaceship modules at another just like it, the first caused the audio alarm on the second unit to go off, presumably to indicate battle damage. I had just observed my very first near-infrared LED, and I was more than excited by the prospects! Better yet, I next discovered that reflected energy from the LED would also trigger the firing spaceship's own alarm, but with a different sound that suspiciously resembled the "collision alarm" on a Navy ship. The *Star Bird* toy contained a fully functional near-infrared proximity sensor! I just had to hack it to create a digital output (versus the audible alarm), and similarly trigger it to fire under computer control.

That done, I installed four of the modified transmitter/receiver units on *ROBART I*, oriented to sense obstacles directly in front of, behind, or to either side. Each sensor covered a cone-shaped region roughly 12 inches out, with a circular footprint at maximum range of approximately 6 inches. The various LED emitters were driven by a square-wave train of 15-microsecond pulses, with a pulse-repetition period of 1.7 milliseconds. A 47-mfd electrolytic and 10-ohm decoupling resistor were configured to supply an extremely heavy current flow (in excess of 2 amps) for the brief on-time, more than enough to destroy the LED under steady-state conditions. The result was an intense pulsed output in a narrow cone, both desirable properties for an object-detection system of this type.

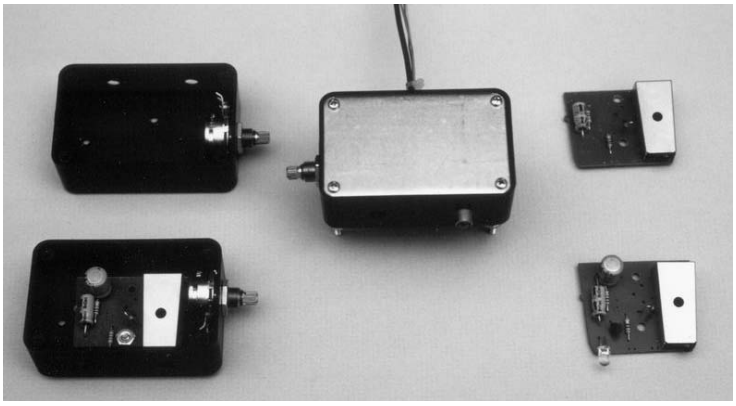


Figure 2-9. Custom-built, near-infrared proximity sensors, housed in plastic Radio Shack® electronics enclosures, are shown here in various stages of assembly.

The receiver employed a *TIL413* photodiode with a built-in filter and lens system. The output of this photodiode was passed through a L/C differentiator network and then

amplified to produce a positive spike for each burst of reflected near-infrared energy detected. These pulses in turn triggered a 555 monostable, which provided an output pulse of approximately 100 milliseconds to ensure sufficient time to invoke the interrupt service routine. These sensors proved to be very effective, and their relative simplicity and low cost made it possible to eventually add six additional units to increase the volumetric coverage. This multi-sensor approach also improved situational awareness by better establishing a detected object's location, as opposed to merely sensing its nearby presence.

Nine of the ten units were placed in the first and fourth quadrants relative to the longitudinal (i.e., path axis) centerline (Figure 2-10), as most movement was in the forward direction. Additionally, when collision-avoidance routines did call for reverse motion, the vehicle generally backed into space just previously vacated, so the odds of a rear impact were greatly reduced. This front-loaded sensing strategy was largely inspired by the biological analog seen in Nature, where evolution in almost all cases provided eyes on the front of the head, with no rear-facing coverage. I would later rethink this sensor-placement philosophy for a number of reasons, but this initial configuration served to get things going.

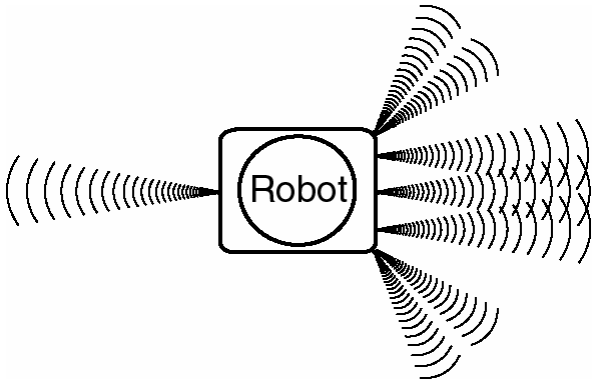


Figure 2-10. Nine of the 10 short-range proximity sensors (two of which overlap vertically) were arranged to provide coverage in the forward hemisphere, with only a single unit covering the rear.

Since horizontal (angular) resolution was more important than vertical, these units were typically mounted in a column to increase their vertical coverage (Figure 2-11), with their outputs hardwired in an OR configuration and treated as a single source. Improvements to the original detection circuitry ultimately provided an increase in maximum range (i.e., from 12 to 30 inches, depending upon surface reflectivity), and potentiometers were installed on each unit to adjust individual sensitivities. I added additional LED emitters to the forward sensors to further increase their range, and to compensate for some blinds spots discovered on either side. The final design performed so well that many of the tactile feeler probes were no longer needed and subsequently removed, leaving only two. More so than anything else, it was probably the availability of these 10 near-infrared proximity sensors that made *ROBART I* such an effective early demonstration of a fully autonomous behavior-based robot.

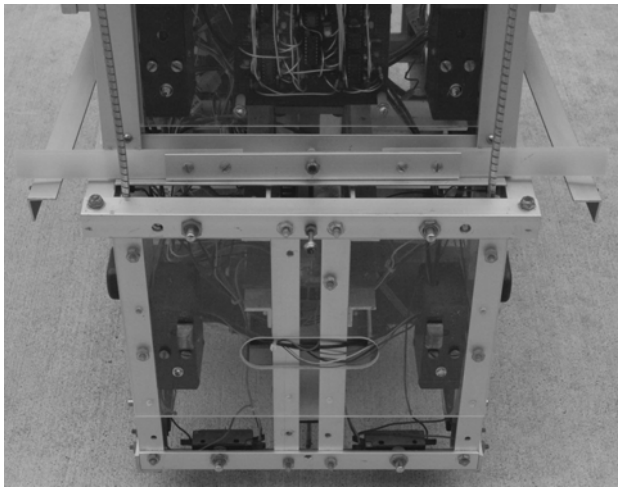


Figure 2-11. Shown here with the front tactile panel removed, the four forward-facing proximity sensors are arranged in stacked columns (left and right) to increase the vertical field of view. Only two of the original feeler probes remain, vertically configured in front of the left and right structural columns (top).

It was while writing the collision-avoidance interrupt service routines that I first ran across a rather non-intuitive disconnect between computer polling speeds and real-time control. This is an interesting subtlety that almost all roboticists have encountered at one time or another, for the human brain is very slow in comparison to a computer, and we tend to think in terms of the former. For example, if both the left and right proximity sensors on the front detect an obstacle, one sensor will trigger before the other almost every time, even if only by a margin of microseconds. In a simplistic case-based collision-avoidance strategy, the computer may read the first one and instantly enact the associated pre-programmed response.

Even if both sensors detect at precisely the same instant, the first conditional that tests positive may preclude ever getting to the next, effectively biasing the results in accordance with the polling order. The interrupt service routine must specifically take this issue into account, perhaps repeatedly polling all inputs with a very short loop delay to be sure another sensor did not also detect. To put things somewhat into perspective here, the fastest human reaction time to visual input is thought to be 120 to 160 milliseconds (Duffy, 2004), so a 100-microsecond delay with 100 loop passes is still an order of magnitude quicker than a professional athlete.

The same situation can also arise when writing an output command and then immediately performing a diagnostic check to ensure compliance. I typically test to see if the drive controller executed a requested motion command by looking for shaft-encoder displacement, then issue a diagnostic-failure report if nothing happens. I learned from experience not to implement this verification step in the very next section of code, due to the small but significant delay associated with the startup inertia of the motor. This seems fairly obvious in retrospect, but is often not intuitive in the beginning.

Scanning Proximity Sensor

I modified the basic proximity-sensor design to yield a head-mounted scanner (Figure 2-12) that could be positioned 100 degrees to either side of centerline, which was

extremely useful in locating open doorways and clear paths for travel. The system used two adjacent LED emitters for increased range, with improved sensitivity and angular resolution provided by a 4-inch parabolic reflector that focused returned energy on the photodiode lens. Although no range information was provided, this approach could establish the horizontal location of both sides of a door opening to within 2 inches of arc at a distance of 5 feet.

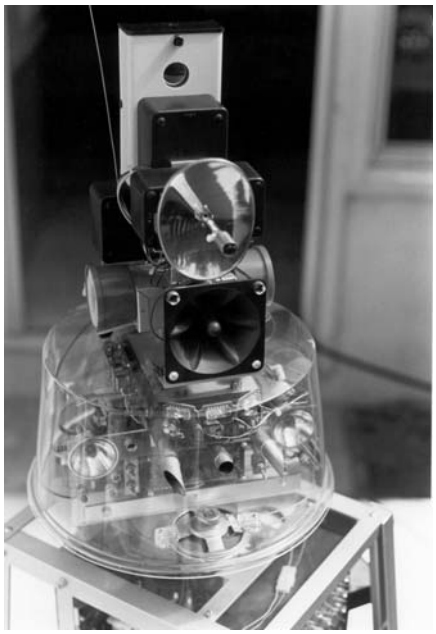


Figure 2-12. The parabolic reflector focused returned energy from two LED emitters onto a PIN photodiode detector for increased range out to about 5 feet, with very good angular resolution.

The receiver circuitry was essentially the same as that employed in the original proximity sensors, but with increased amplifier gain and an adjustable threshold detector. Additionally, the 555 pulse stretcher was made retriggerable to provide a constant high output as long as reflected energy bursts were received, free of intermediate resetting. This modification allowed the device to be panned in search of a wall opening, for example, with a low output arising only when reflections stopped, and not momentarily each time the 555 clocked out and reset.

Drive-Overload Sensor

The drive-motor-overload sensor was a last-resort detector, generating an interrupt that reversed the drive wheel and assigned a semi-random steering command in hopes of clearing the obstruction. Upon completion of this pre-programmed evasive action, the original drive and steering parameters were restored, and the robot proceeded as before. I envisioned this sensing modality primarily as a safety check when docking with the battery charger, since other reactive strategies (i.e., proximity and tactile detection) would be disabled during the terminal stages of docking. As it happened, this was a bit of overkill, but since implementation was so trivial, I kept it as an option. It turned out to be more useful whenever the robot tried to climb over small obstacles “below the radar” of normal collision-avoidance defenses.

2.1.3 Biological Inspiration

In looking back over the years, I have noticed most roboticists tend to fall into one of two camps: (1) those who pay no attention whatsoever to Nature's elegant examples; and (2) those who go overboard in strict adherence to the age-old axiom that "Nature knows best." I once worked with an engineer who firmly believed anything and everything robotic should be an exact replica of its human counterpart. No matter that the robot looked like a HMWWV, if there were microphones listening for acoustical signatures from the surrounding environment, they would be mounted in latex structures molded from a human ear (Figure 2-13). I used to joke that if this individual were to design a new torpedo for the submarine community, he would put a pair of flippers on the back instead of a high-speed propeller.

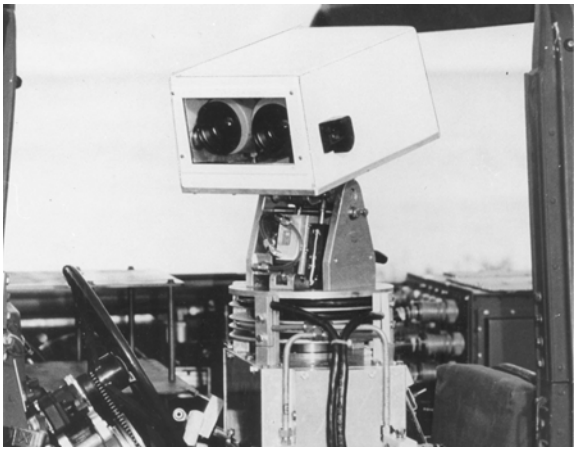


Figure 2-13. The stereo microphones on either side of the robot's head were mounted in latex "ears," presumably to enhance their acoustical performance.

Personally, I believe the optimal approach is a synergistic blending of both strategies. The human body is by far the most complex and sophisticated instantiation of an autonomous system ever to exist, at least in that part of the universe we have experienced to date. (I used to have a cartoon on my office wall that tellingly proclaimed, "Man is the most versatile servo-mechanism mass-produced by unskilled labor," until it was sadly confiscated in the early nineties by the PC Police.) So clearly we have a lot to learn from biological precedents. On the other hand, if one of the biggest selling points for robotics in general is an ability to perform tasks that humans cannot, would we not be seriously impeding progress to insist on faithfully patterning everything after humans? There are pros and cons associated with both biological and artificial existence, and greater success would seem more likely to follow a careful exploitation of the advantages of both.

Consider the earlier case regarding optimal placement of near-infrared proximity detectors, a sensor modality for which there is no obvious biological analog in humans. We have a sense of touch, which can to some extent be replicated on a robot in the form of tactile bumpers or feelers. Our sense of hearing can reasonably be emulated with microphone arrays and sophisticated signal processing. Similarly, the sense of sight maps into an artificial counterpart in the form of computer vision, using monocular or stereo cameras with appropriate image processing. Smell is less obvious, but even *ROBART I* had both a smoke detector and a Figaro gas sensor (Everett, 1982), as did

ROBART II (Everett, 1985). More sophisticated olfactory sensing has since evolved, including a path-following robotic capability to track a scent along the floor (Deveza, et al., 1994; Bains, 1994; Russel, et al., 1994), and continued development of artificial noses in general (Keller, et al., 1996; Gardner & Bartlett, 1999; AAAI, 2005).

But to the best of my knowledge, humans do not directly sense proximity per se, which is not to say it cannot be inferred to some extent from the other sensor modalities listed above. For example, humans do have an amazing (although scarcely known) ability to echo-locate, similar in many respects to the more sophisticated sonar capabilities that whales, dolphins, bats, and even some species of birds rely upon to navigate and find food. We likewise have an acoustic sense of our volumetric surroundings, which is a combination of not only the sounds generated by objects (including ourselves) in that environment, but also their echoic interaction with reflective surfaces (Schwitzgebel & Gordon, 2000).

It has been known for quite some time that blind people can very effectively exploit this phenomenon to avoid obstacles in unfamiliar territory. Daniel Kish, who is completely blind, can derive sufficient spatial information on his surroundings to safely ride a bicycle, simply by clicking his tongue and listening to the returning echoes (Wiki, 2005). Recent studies have shown that human echolocation is more effective in judging distance to a stationary obstacle if the echolocator (i.e., tongue clicker) is in motion, as opposed to standing still (Rosenblum et al., 2000). Rather than a brain-plasticity phenomenon, echolocation is a natural process that can be mastered by sighted individuals with minimal training (Schwitzgebel & Gordon, 2000).

All things considered, I do not think we will get too far in the business of designing intelligent machines unless we take advantage of everything that both nature and technology have to offer. After all, technology is nothing more than another small step in the ongoing march of evolution (Kurzweil, 1999), although current limitations leave much to be desired when it comes to replicating biological functionality. Computer vision has a long way to go, for example, to be competitive with our own amazing capacity for visual perception, just as the artificial nose falls far short of our olfactory discrimination. And in both cases, these reasonably impressive human capabilities are even further outstripped by the animal kingdom. It is estimated that a dog's nose can distinguish 20 million times as many separate odors (Churchland, 1995), while night vision, barely functional in humans, is a key factor in the survival of many animal species.

Humans have certain sensor capabilities that are fairly well understood, others which are not so well known, and perhaps even a few more we have not yet even discovered. The significant point to note here is that modern technology for the most part cannot come close to matching the performance of these biological counterparts. For this reason, the argument that "humans do not need it, so why should robots?" is insufficient cause to reject an artificial non-human-like sensor modality, or even a different scheme of employment for a human-like sensor. Until artificial instantiations can stand head-to-

head with human sight, hearing, touch, and smell, we will need to level the playing field a bit by adding things robots do well that humans perhaps cannot do at all.

2.2 Navigational Behaviors

The software dealing with all this tactile, proximity, and range data was divided into two categories: (1) the main program, and (2) interrupt service routines. The main code handled navigational control of the robot as required to proceed from place to place in the accomplishment of a desired task or goal (i.e., the so-called *deliberative behaviors*). All information available from whatever source was used to this end, and navigational routines were written in loop form to facilitate repetitive polling, with the appropriate exit requirements built into the loop. Loops could be cascaded as needed in the execution of *complex deliberative behaviors*, which will be further discussed in several later sections. In the meantime, we take a closer look at the *reactive behaviors* associated with collision-avoidance interrupt routines.

2.2.1 Reactive Behaviors

Reactive control refers to a behavior-based strategy that directly couples real-time sensory information to motor actions without the use of intervening symbolic representations that model the robot's operating environment. Ron Arkin (1992) lists the following general characteristics of *reactive control*:

- It is typically manifested by decomposition into primitive behaviors.
- Global representations of the robot and its surroundings are avoided.
- Sensor decoupling is preferred over sensor fusion.
- It is well suited for dynamically changing environments.

Robotically speaking, the simplest reactive-control behavior is perhaps illustrated by the basic *Wander* routine implemented by several research groups in the eighties (Everett, 1982; Brooks, 1986; Arkin, 1987; Anderson & Donath, 1998). As used here, the term *Wander* describes a behavior primitive that involves traveling more or less in a straight line until an obstacle is encountered, altering course to avoid impact, then resuming straight-line motion. *Wander* as implemented on *ROBART I* was based on the last four elements of the six-level scheme of proximity and impact detection outlined below:

- The near-infrared proximity scanner mounted on the head.
- The forward-looking *LM1812* sonar.
- Ten near-infrared proximity detectors to sense close obstructions (<18 inches).
- Projecting "cat-whisker" feelers to detect pending collisions (<6 inches).
- Contact bumpers to detect actual impact.
- The drive-motor overload sensor indicating a stall.

The first two categories can be classified as *medium-range non-contact sensors* that looked out ahead of the robot, more in support of higher level deliberative planning behaviors to be discussed later. The next three were considered *close-in proximity and tactile sensors* that required immediate action, and were directly associated with the reactive collision-avoidance behaviors under discussion. Drive-motor overload was a

last-resort internal sensor in case contact with an object was not avoided by any of the above.

The high-priority proximity and tactile sensors monitoring the close-in environment were read by the microprocessor's interrupt-request service routine. Unless deactivated by the main program, the interrupt software would redirect the motion of the robot in accordance with preprogrammed responses specifically tailored to the individual sensor states. An object entering the field-of-view of a proximity sensor on the right, for example, would trigger a slight heading change to the left, and vice versa. On the other hand, a preprogrammed response for a right-front-bumper impact would be a little more complex, consisting of the following steps:

- Stop all forward travel.
- Turn steering full right.
- Back up for *x* number of seconds.
- Stop and center steering.
- Resume forward travel.

The above example clearly illustrates Arkin's "decomposition-into-primitive behaviors" characterization, the specific primitives in this case (i.e., for bumper impact) being *stop*, *turn*, *move backward*, and *move forward*. *Wander* itself may be considered a fuzzy step up the hierarchy as shown in Table 2-1, residing at an intermediate level between the purely *reactive behaviors* below and the more *deliberative behaviors* above. In other words, *Wander* would *deliberately* try to keep moving forward, *reacting* if necessary to either close proximity or actual impact.

LEVEL	BEHAVIOR	RESULTING ACTION
Higher	(Future Additions)	Deliberative navigational strategies
Intermediate	*Wander*	Seek clear path along new heading
Lower	*Get Unstuck*	Escape a trapped condition
	Proximity Reaction	Veer away from close proximity
	Impact Reaction	Veer away from physical contact

Table 2-1. The navigation scheme on *ROBART I* envisioned a three-layer hierarchy of behaviors, with higher level *deliberative* routines supported by lower level *reactive* collision avoidance and detection.

The ideal situation would have the advanced planning executed by the deliberative navigational loop be so effective as to make interrupt generation by close-in contacts a rare occurrence, but this would require more numerous long-range sensors than initially affordable. I made a point of stating in my thesis that this interrupt-driven method of collision avoidance was not necessarily preferable or recommended for future systems of greater sophistication; it merely lent itself well to low-budget applications involving a minimal number of microprocessors (Everett, 1982). One rather significant problem with my initial implementation, however, was that it was an either–or situation in terms of execution: the *deliberative behaviors* were put on standby when the *reactive behaviors* took over.

Conversely, human *conscious* and *subconscious* behaviors (which are what we seek to emulate here) run in parallel, each for the most part unimpeded by the other. So, a much better approach would involve the multi-tasking of concurrent processes versus suspension of one to attend to the other. Unfortunately, multi-tasking operating systems were yet to arrive upon the scene, and writing your own was a bit beyond my skill set in 1981. To achieve some degree of parallel processing, I relied upon dedicated hardware controllers for mobility (i.e., drive and steering) and head function (i.e., scanning and tracking). I would attempt to further compensate in this fashion on *ROBART II* (1982–1992) by increasing the number of onboard microprocessors from two to thirteen.

2.2.2 Reactive Trap Scenarios

The forward-looking proximity detectors were set for a maximum range of 18 inches to give sufficient time for course correction, while the side sensors could see out to a distance of only 12 inches, since less time was needed to react. The objective was to keep the protection envelope as small as possible without compromising performance, thus facilitating passage down narrow hallways, through doorways, and between obstructions. Too large a detection range substantially reduces a vehicle's perceived clear space wherein it can navigate, with the result that much time could be spent turning circles in the open space of a room, unable to exit through a door.

Even with reduced side ranges, situations were sometimes encountered that left the robot "boxed in," finding itself trapped between multiple obstacles in congested surroundings. Interrupts generated by proximity returns on both sides at once could hinder an orderly exit from this predicament, particularly in the case of a tricycle-drive configuration that could not pivot in place. The robot would oscillate back and forth in a repetitive turn-left/turn-right fashion, essentially going nowhere. What was needed was some further means of "drawing in" the protected envelope until clear of this tight spot, and there were a number of ways this could be done. I chose the simplest solution: temporarily ignore the near-infrared proximity detectors until out of the jam.

Less obvious was the problem of how to detect the situation in the first place (i.e., how could the robot determine it was boxed in?). The first indication would be an excessive number of interrupts triggered by proximity returns, tactile feelers, or possibly even bumper impacts. This "interrupt count" could be easily calculated by incrementing a register (*irq.fre*) whenever a collision-avoidance interrupt occurred, and periodically clearing this register after some predetermined amount of time. This *count-reset function* was performed by the real-time-clock routine each time the minutes register incremented, which was probably not the optimal solution, but it worked well enough. Thus, if the *irq.fre* register value ever exceeded a specified limit, then the interrupt frequency had exceeded that same count in roughly the last 60 seconds.

A second piece of information was needed to verify a trapped condition did indeed exist, as a large number of interrupts could easily arise from side returns when the robot was navigating a hallway, without the vehicle being trapped. Further observation revealed that tight situations almost invariably were accompanied by rear-bumper impacts, which otherwise seldom occurred. So a combination of too many interrupts plus

rear-bumper contacts was used as the governing conditional, and the IRQ service routine responded by disabling the near-infrared proximity detectors. (I later added repetitive alternating left-right detections as a third conditional trigger.) This shrinkage of the protected envelope allowed the robot more maneuvering room, relying on tactile sensors alone for guidance as it attempted to free itself from the trap. The proximity sensors were later re-enabled by the main navigation software after the congested area was cleared.

2.2.3 Reactive Behavior Modification

Another fundamental problem with canned *reactive behaviors* is that they can annoyingly interfere with the objectives of any high-level *deliberative behavior* in execution at the time the low-level reactive routine is asserted. For example, assume the navigational software is trying to go through a doorway but the proximity sensors keep seeing the doorframe on one side or the other. If the opening is sufficiently narrow to where both sensors simultaneously perceive an obstacle, the robot will not be allowed to pass through. The reactive avoidance behaviors are responding in unyielding fashion to a fixed-threshold stimulus, fully oblivious to the higher level goal to travel through the open doorway. The robot will eventually interpret this as a trapped condition and seek to escape.

Accordingly, the aforementioned concept of suppressing the *low-level reactive behaviors* (i.e., by disabling the proximity sensors) to assist in escaping a trapped condition was an important milestone in the early evolution of *ROBART I*. As human beings, our natural response to a sharp object (i.e., cactus barb) impaling our tender flesh is to instinctively pull away, yet we intentionally suppress this very same *reactive behavior* when getting a flu shot. Our *conscious deliberative behavior* at times must alter our *subconscious reactive responses* to achieve the desired goal, and a robotic system is no different in this regard. The robot's *deliberative behavior* must similarly assert itself under certain conditions by altering the "fixed-threshold canned-response" mentality of the *reactive behavior*, or be forever frustrated. One obvious way to do this would be disabling the *reactive behavior* altogether, as we have seen.

An alternative option was disabling only the interrupt capability of these sensors, rather than deactivating them altogether, which achieves the desired *behavior suppression* while preserving access to the sensor data itself. Judiciously polling the proximity sensors, for example, could conceivably result in more intelligent maneuvering out of the trap, relative to the rather primitive "bump-and-recover" routines supported by tactile feedback alone. An even better plan would have called for first reducing the detector sensitivity to draw in the protected envelope, followed by disabling the interrupts altogether as a final resort. This improvement could have been implemented as a minor hardware change to the threshold detector circuitry, but that upgrade was postponed for installation on *ROBART II*. As a military grad student, I had to stay on schedule.

2.3 Summary

The concept of selectively disabling or modifying the reactive collision-avoidance software became a critical aspect of the underlying philosophy for interaction with future additional layers of *intermediate* and *high-level deliberative* behaviors. In addition, the

contents of certain registers could be changed by the interrupt routines to alter the planning processes of the main program after a return from interrupt. This bidirectional approach provided the necessary communication between the interrupt routines and the main program for more intelligent *reactive* and *deliberative* behavior coordination.

One of the first *deliberative behaviors* incorporated on *ROBART I*, searching for and docking with its battery charger, effectively illustrates this important concept. Instead of instinctively triggering an avoidance maneuver at some pre-established minimum-distance threshold, the forward-looking sonar should be monitored for appropriate range closure commensurate with the homing procedure. This stand-off information would help the system tell when a proximity sensor detected the recharging station itself, so that the interrupt software would not try and steer away from the "obstacle" with which the robot was trying to connect. And just prior to bumper contact, the *Docking* behavior had to disable even the fall-back tactile reaction, as will be discussed in the next section.

2.4 References

Anderson, T.L., and Donath, M, "Synthesis of Reflexive Behavior for a Mobile Robot Based Upon a Stimulus-Response Paradigm," SPIE Mobile Robots III, Vol. 1007, W. Wolfe, Ed., Cambridge, MA, pp. 198-211, November, 1998.

Arkin, R.C., "Motor-Schema-Based Navigation for a Mobile Robot: An Approach to Programming by Behaviors," IEEE International Conference on Robotics and Automation, Raleigh, NC, 1987.

Arkin, R.C., "Behavior-Based Robot Navigation for Extended Domains," *Adaptive Behavior*, Vol. 1, No. 2, MIT Cambridge, MA, pp. 201-225, 1992.

Armstrong, D.M., "What is Consciousness," *Nature of Mind and Other Essays*, Cornell University Press, Ithaca, NY, 1981.

Bains, S., "Robots Mark Their Territory," *SPIE OE Reports*, No. 124, pp. 1, 9, April, 1994.

Brooks, Rodney A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, April, 1986.

Churchland, Paul M., *The Engine of Reason, the Seat of the Soul: A Philosophical Journey into the Brain*, ISBN 0-262-03224-4, MIT Press, Cambridge, MA, 1995.

Deveza, R., Russell, R.A., Thiel, D., and Mackay-Sim, A., "Odour Sensing for Robot Guidance," *International Journal of Robotics Research*, 1994.

Dretske, Fred, "Perception Without Awareness," *Perceptual Experience*, Tamar Gendler & John Hawthorne, Eds., Oxford University Press, Oxford, UK, 2005.

Duffy, K., "Reaction Times and Sprint False Starts," downloaded from following website: http://condellpark.com/kd/reactiontime.htm, April 3, 2004.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot*, Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982.

Everett, H.R., "A Second Generation Autonomous Sentry Robot," *Robotics Age*, pp. 29-32, April, 1985.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995.

Frederikson, T.M., and W.M. Howard, "A Single-Chip Monolithic Sonar System," *IEEE Journal of Solid State Circuits*, Vol. SC-9, No. 6, pp. 394-402, December, 1974.

Gardner, J.W., and Bartlett, P.N., *Electronic Noses*, Oxford University Press, Oxford, UK, 1999.

Keller, P.E., Kangas, L.J., Liden, L.H., Hashem, S., and Kouzes, R.T., "Electronic Noses and their Applications," Proceedings, World Congress on Neural Networks, San Diego, CA, pp. 928-931, 15-18 September, 1996.

Kurzweil, Ray, *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*, ISBN 0-670-88217-8, Penguin Books, New York, NY, 1999.

Lycan, William G., and Ryder, Zena C., "The Loneliness of the Long-Distance Truck Driver," *Analysis 63*, pp. 132-36, 2003.

Rosenblum, L.D., Gordon, M., and L. Jarquin, "Echolocation Distance by Moving and Stationary Listeners," *Ecological Psychology*, Vol. 12, No. 3, 2000.

Russell, R.A., Thiel, D., and Mackay-Sim, A., "Sensing Odour Trails for Mobile Robot Navigation," Proceedings, IEEE International Conference on Robotics and Automation, San Diego, CA, Vol. 3, pp. 2672-2677, May, 1994.

Schwitzgebel, Eric, and Gordon, Michael S., "How Well Do We Know Our Own Conscious Experience? The Case of Human Echolocation," *Philosophical Topics*, Vol. 28, No. 2, Fall, 2000.

Spitz, Herman H., *Nonconscious Movements: From Mystical Messages to Facilitated Communication*, ISBN 0-8058-2563-0, Lawrence Erlbaum Associates, Mahwah, NJ, 1997.

Synertek, *SY6500/MCS6500 Microcomputer Family Programming Manual*, Synertek Systems, Santa Clara, CA, August, 1976.

Wiki, "Animal Echolocation," http://en.wikipedia.org/wiki/Animal_echolocation, January 4, 2005.

3
Automatic Recharging

"Estimated amount of glucose used by an adult human brain each day, expressed in M&Ms: 250."

Harpers Index
October, 1989

For an autonomous robot to be useful when operating in unpopulated, remote, or hazardous environments, I reasoned, it must be able to support itself to a large degree, particularly in recharging its own battery. Detecting the need to recharge was rather trivial, so the problem became how to locate and connect to a free-standing docking station. An extremely reliable process was required, one not rendered ineffective by unforeseen obstacles or changes in the environment, and without a complexity that overshadowed the rest of the system. Finding the charger could presumably be accomplished with some type of head-mounted sensor, so the real engineering challenge would be in making the electrical connection.

The dual requirements for simplicity and reliability effectively ruled out the standard humanistic approach of trying to align a special plug on the robot with a mating receptacle on the charger. While this ultimately could have been done, it would require rather sophisticated hardware as well as software, and was thus susceptible to a myriad of complications. What I needed was a method that was independent of the direction of approach, required no precision alignment procedures, and provided a good electrical connection every time. To meet these criteria, I chose the vertically symmetric contact configuration depicted in Figure 3-1 below.

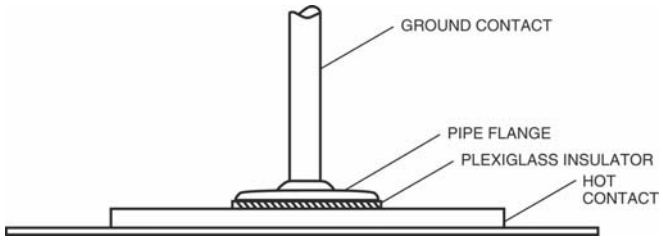


Figure 3-1. The base of the battery-recharging station presented a symmetrical contact configuration that was independent of the robot's direction of approach.

With the electrical interface thus conceptualized, the question became how to find and then home in on the charging station. It is interesting to look for parallels here in nature, since evolution has provided many robust tracking solutions for a wide variety of fairly simplistic life forms. Homing pigeons, for instance, fly back to their nest from distances of hundreds of miles using a combination of celestial and magnetic navigation. One theory is that trace particles of magnetite in the pigeon's brain structure are influenced by the Earth's magnetic field, creating a biological compass (Kreithen, 1983; Churchland, 1995). As another example, adult salmon retrace rather complicated paths through branching streams and white-water rapids in a determined journey to lay their eggs in the place of their own birth, relying on a keen sense of smell to guide them back to the precise location.

These biological instantiations of long-range homing, however, are far more complex than anything required for this application, which by comparison is a relatively straight-forward problem of moving towards a nearby object already viewed by onboard sensors. A more appropriate analogy, in the form of a bottom-feeding crab searching the seafloor for food (Figure 3-2), is presented by Paul Churchland (1995) in his fascinating philosophical portrayal of the brain: *The Engine of Reason, the Seat of the Soul.* The foraging crab is an especially fitting parallel for *ROBART's* desired homing behavior, in that (1) both systems are seeking an energy source, (2) the source can be approached from any angle in the horizontal plane, (3) both employ sensors scanned only in azimuth to locate their target, and (4) both use the resulting sensor-angle information to drive a motor response for converging upon the perceived location.

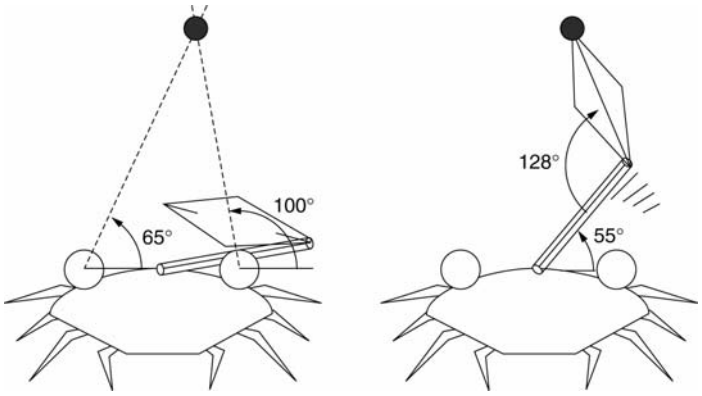


Figure 3-2. The crab's rather primitive eyes converge upon a food source (left), generating *sensor-input vectors* that in turn are *transformed* by the neural network of its brain, producing *output-control vectors* to position the arm/claw appendage (right) for intercept (adapted from Churchland, 1995).

Evolution has provided the lowly crustacean with precisely those cognitive skills supportive of survival, and presumably nothing more. Being a bottom-feeder, the crab's "hand-eye coordination" needs are greatly simplified by restriction to the horizontal plane. The crab's offset eyes have no up–down tilt capability, but rotate independently about their respective vertical axes, thus enabling triangulation of the perceived food-source coordinates as seen in Figure 3-2. It also has a two-degree-of-freedom arm/claw manipulator to grasp the food, where the claw position is determined by the elbow and shoulder joint angles, similarly constrained to planar motion. (This plane of restriction, however, is referenced to the crab's body, which to some extent can be tilted with respect

to the sea floor by its leg motions, thus providing some limited 3-D reach.) All that is required for effective *sensorimotor coordination* is a transformation of *sensor input vectors* representing the scan angles (i.e., of the two eyes) into *motor control vectors* which yield the joint angles for arm and claw (Churchland, 1995).

But what exactly does all that mean? The terminology *vector coding* (Churchland & Sejnowski, 1992) describes, among other things, the manner in which nature encodes sensor data for subsequent cognitive processing by the parallel network of neurons that make up the brain. The concept was first championed by Karl Pribram (1971), but was overshadowed by the more popular detector-cell theories of the time (Bridgeman, 1993). As implied in the preceding paragraph, the concept of *vector coding* also applies to the brain's outgoing signals that direct appropriate motor responses, thus greatly facilitating the *transformation* of input conditions to output action.

Probably the most familiar example of vector coding is illustrated by the three primary colors (i.e., red, yellow, and blue), which can be combined in various ratios to produce any hue imaginable (Figure 3-3). The efficiency is obvious, if one compares this method of specifying the respective weights for just three variables to the alternative of having to uniquely identify each and every possible color combination as a separate entity. Not surprisingly, the retina contains three types of cone-shaped receptors, each tuned to a different wavelength of the optical spectrum, and the stimulation levels of these receptors collectively define the perceived color of incident radiation.

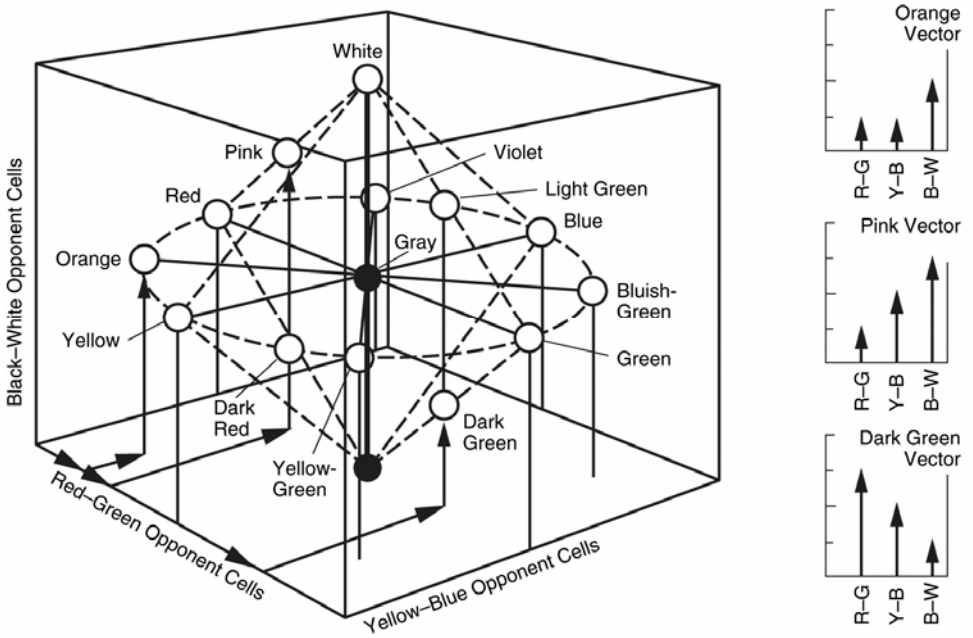


Figure 3-3. In the vector-coded human color space, the various hues are distributed along a continuous circle around a central vertical axis, where vividness is a function of radial distance (adapted from Churchland, 1995).

Another such example is seen in our highly developed sense of taste, but there are four receptors at work here instead of just three, discriminating sweet, sour, salty, and bitter. The combination of weighted scores assigned to these four categories provides the

necessary encoding for a precise accounting of the taste of any substance coming into contact with the tongue. This collective pattern of activation across all four receptors forms a unique and repeatable "signature" for the way something tastes, conveyable to the brain as four weighted scores. This *combinatorial system of representation* employed by *vector coding* supports a very fine-grain analysis of the subtleties associated with a particular sensory experience (Churchland, 1995). In contrast, imagine the difficulty in trying to provide this same descriptive information regarding almost imperceptible differences in taste through the medium of spoken language!

How then does this robust scheme of *vector coding* relate to the previous example involving our hungry crab? Referring again to Figure 3-2, the crab has two visual receptors, and their inputs to the neural network making up its brain consist of their respective scan angles to the perceived food. Assume input values of 65 degrees for the left eye and 100 degrees for the right produce the *excitatory neural connections* shown as solid lines in Figure 3-4 below, and the *inhibitory connections* shown as dotted. The network transforms this sensor input state to an appropriate output state that commands the shoulder joint to the 55-degree orientation, and the elbow joint to 128 degrees, enabling the claw to grasp the food. Another way of looking at this is that the *transformation* is a learned "look-up table" of arm/claw-joint angles corresponding to the set of all possible spatial locations as seen by the eyes. The objective, of course, is to make the claw converge on that particular point in the feeding plane where the crab's eyes perceive the food.

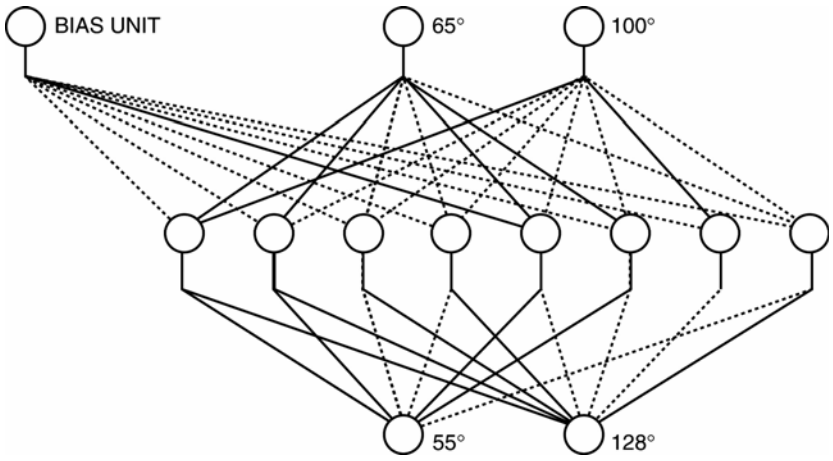
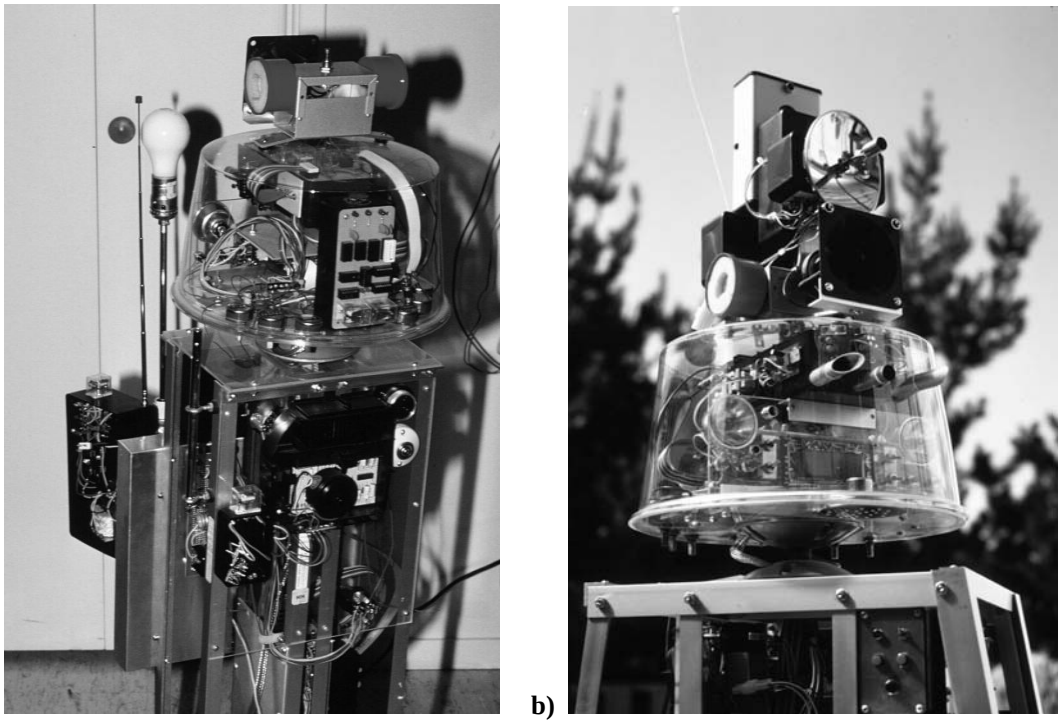


Figure 3-4. Scan angles associated with the crab's converging eyes (top) provide a vector-coded *input representation* of spatial position, which is *transformed* by the network to produce a vector-coded joint-angle *output solution* (bottom), enabling the claw to seize the food (adapted from Churchland, 1995).

In actuality, this crab example is somewhat of an over-simplified *feed-forward network* that ignores the temporal issues required for stable dynamic control (velocity feedback, for example), and therefore representative of what is commonly known as an open-loop "bang-bang" controller. Any unfortunate crab so handicapped would quickly starve to death, unable to effect the rapid yet smooth arm/claw motion (i.e., without serious overshoot) necessary to quickly snatch its prey. After pointing this out, Churchland proceeds to describe a more complex neural arrangement involving *descending* or *recurrent pathways* that provide the necessary feedback.

3.1 Beacon-Tracking System

The task of locating *ROBART's* recharging station could have been accomplished by any of several methods, but the requirement for high resolution over fairly short ranges made an optical tracking system a prudent choice. I used an ordinary incandescent lamp for the homing beacon, situated at the same height as the scanning optics on the robot's head as shown in Figure 3-5a. (This choice was driven by the need to minimize development time and cost, and my previous experience with this same approach on *CRAWLER I*.) The tracking sensors were simply three photocells, each with a 12-inch collimating tube to improve angular resolution, arranged in a horizontal array. The tubes were aligned in a diverging configuration, with the center tube parallel to the forward axis of the head as shown in Figure 3-5b.



a) **b)**

Figure 3-5. a) A simplistic three-element photocell array was used to track the 60-watt incandescent bulb atop the recharging station; **b)** the three collimating tubes for the sensor array are seen projecting from the plexiglass cover just above the headlamps and clock display.

To initiate a homing sequence, the robot would activate the beacon via a radio link, track its perceived location in azimuth with the head-mounted photocell array, and then home in by adjusting the steering angle as a function of beacon-offset relative to centerline. I chose to implement this behavior in dedicated hardware that controlled the head position, located on the *Optical Board* (Figure 3-6). This circuitry allowed for three different control modes, referred to as (1) *Scan Mode,* (2) *Track Mode,* and (3) *Position Mode.* A brief explanation of these three modes follows, with additional detail provided in Appendix A.

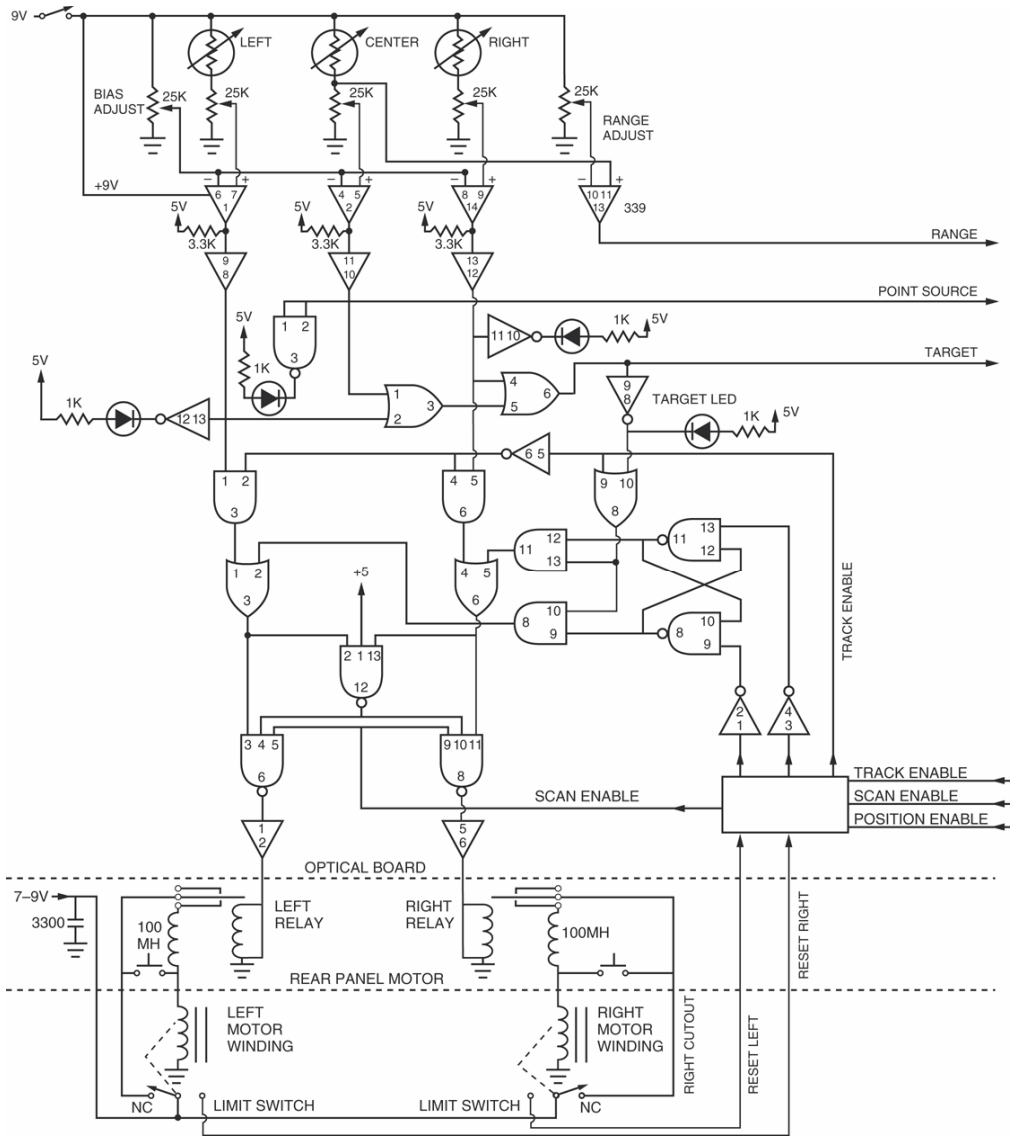


Figure 3-6. The head-pan-axis motor was controlled by the *Optical Board*, while *Interface Board Number 5* evaluated the computer commands in conjunction with other inputs to determine the control mode: *Scan, Track, or Position.* Note similarity to neural network depiction of Figure 3-4.

In *Scan Mode,* the head would automatically pan back and forth between ±100 degrees of centerline, seeking a point source of light with sufficient intensity to trigger the photocell comparators. This scanning action was controlled by the scan flip-flop, set and reset by limit switches at both extremities of pan travel, causing the motor to reverse direction each time and scan the other way. If any of the three comparators indicated beacon acquisition, the scan flip-flop was gated out, and the tracking inputs took over motor control.

In *Track Mode*, the head would servo off the homing beacon situated on the recharging tower. The system would actually track any bright light source, so it was up to the software to ensure that this source was the beacon. The tracking process was initiated automatically by a low-battery condition, through alteration of the robot's

behavior-selection procedure in such a way as to terminate the routine in progress (see Section 4).

In *Position Mode*, the head would seek whatever position the *SYM* wrote to a four-bit latch, as discussed in Section 1. A 7485 four-bit comparator compared the command stored in the latch with the output of the head-position A/D converter and servoed the head in the same way previously described steering circuitry positioned the drive wheel. Thus the head could be made to center itself after a scan operation, or to seek any of 16 fixed positions on command.

In 1981, I obviously had no knowledge or experience with the concepts of neural-network-based sensorimotor control schemes and/or vector coding. My background in the more conventional field of microprocessors was almost as bleak, limited to a few months of unsupervised trial-and-error learning experiences with 6502 assembly language. This lack of familiarity with computers in general biased me towards a dedicated-hardware logic design, which in retrospect was probably more closely aligned with the neural-network approach illustrated by Churchland's famous crab. Let us now consider such a neural-net parallel (see Figure 3-7) to the logic circuitry presented earlier in Figure 3-6.

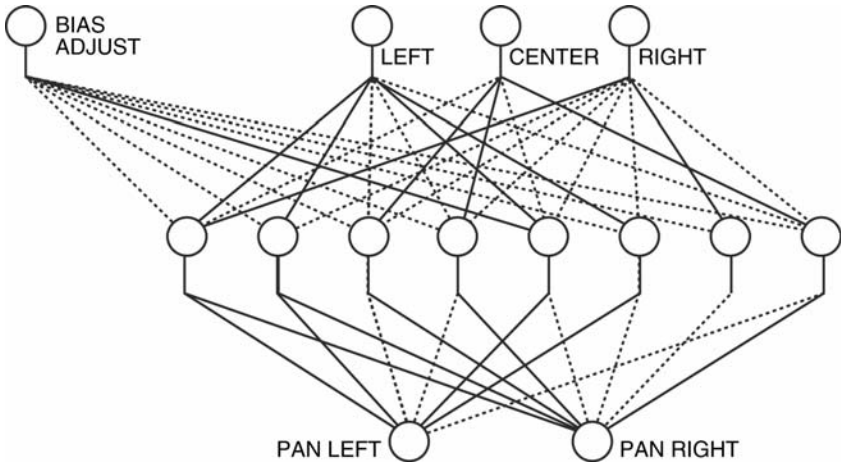


Figure 3-7. In this neural-net analogy to the *Optical Board* logic circuitry for controlling the head, the three photocells at the top provide inputs that determine the head-pan-motion outputs at bottom. Note the absence of any feedback loops, as was also the case in the hardware version presented in Figure 3-6 (pan velocity was fairly slow and well damped).

To keep it simple, we will initially address only the *Track Mode* functionality of the *Optical Board*, ignoring *Scan Mode, Position Mode*, and the homing behavior itself. As such, we are essentially depicting a model of the crab's eyes tracking the desired food item. The input vectors are derived from the three-element sensor array, and are binary weighted in that the associated comparators (Figure 3-6) transition from a *low* to *high* state when their analog input voltages (i.e., photocell outputs) exceed their reference inputs. Note that this is one hierarchical level below that of Churchland's network for positioning the arm/claw as a function of eye angles, which corresponds instead to *ROBART* adjusting steering angle as a function of head angle during the homing behavior. We will get to that example later (Figure 3-9).

Meanwhile, referring again to the schematic of Figure 3-6, the tracking inputs to the optical board circuitry came from the left and right photocells in the array. Their respective comparator outputs indicated a greater light intensity on either side of center, referenced to the middle photocell. The appropriate motor winding was energized so the head would turn to regain maximum intensity at the center, thus tracking the source. All this happened only if at least one of the photocell outputs was above an adjustable set-point (supplied by the *Bias Adjust* potentiometer); otherwise, the system reverted to *Scan Mode* and searched for a bright light source. Any comparator output signaling intensity above the set-point gated out the automatic scan, whereupon the tracking inputs took over.

When the photocell outputs indicated the head was correctly positioned (i.e., pointing at the source), the pan motor winding was de-energized. Once the computer detected this condition (i.e., *Target Output* high), it would interrogate the beacon to verify that it was indeed the observed source. (I very prudently added this additional step after the robot tried to "mate" with my ex-wife's much-prized antique table lamp (Figure 3-8), while she unfortunately was right there watching.) Verification was accomplished by setting *Scan Enable* low, then turning off the beacon via the radio link while checking to see if *Target Output* went low. An incorrect source would keep *Target Output* high.

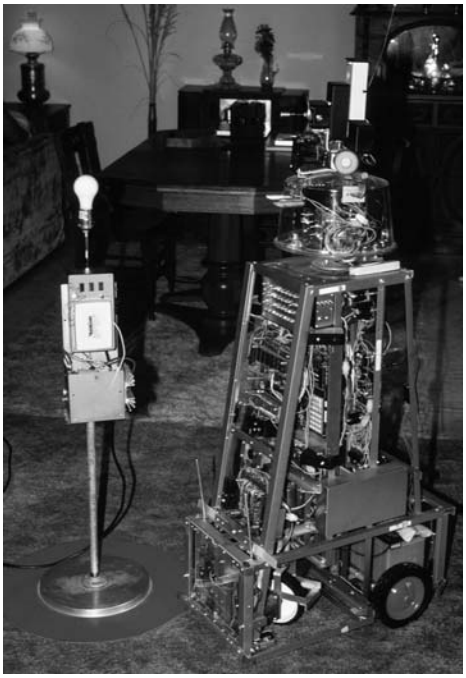


Figure 3-8. *ROBART I* had to discriminate between the homing beacon on its recharging station and any other source (such as the antique table lamp shown in upper left corner) situated at the same height.

If the observed illumination was not the beacon, the computer set *Track Enable* low so the incorrect source would be ignored, set *Scan Enable* high to reinitiate the scan, and waited as the head panned for the incorrect source to clear (*Target Output* going low). As soon as this happened, *Track Enable* was set high again, and a new source was sought. The process repeated until the correct source was found, until three incorrect

sources had been interrogated, or until a specified time limit had elapsed. Either of the latter two conditions indicated that the beacon was not in the immediate scan field (first and fourth quadrants relative to centerline), whereupon the robot would perform a Y-turn and check to the rear.

Once the beacon had been located and its bearing and relative range announced through speech synthesis, the robot began to home in on the charger. The *SYM* would repeatedly read the head position, representative of the bearing to the charger, and send an identical command to the steering motor. The head automatically tracked the light source as the robot turned, causing the relative bearing to the beacon to decrease, which caused the software to reduce the turn angle. This process continued in loop fashion until eventually the beacon was directly in front of the platform, at which point the drive wheel would be centered with the robot moving directly towards the charger.

I did not realize it at the time, but in implementing this beacon-homing behavior I had inadvertently copied a very important evolutionary concept concerning intelligent *sensorimotor coordination* (i.e., controlling the behavior of a system in direct response to its current perception of the surrounding environment). As stated by Churchland (1995):

> "If the external environment is represented in the brain with high-dimensional coding vectors, and if the brain's intended bodily behavior is represented in its motor nerves with high-dimensional coding vectors, then what intelligence requires is some appropriate or well-tuned transformation of sensory vectors into motor vectors!"

Relative to the two-degree-of-freedom crab, *ROBART's* sensorimotor-coordination problem was even simpler since there was but a single angular variable for sensor input and another for steering output. Accordingly, the required input-to-output *transformation* was nothing more than *drive steering angle* equals *head pan angle* (see Figure 3-9). To further illustrate the incredible power of this vector-coding approach, we will later examine a situation where a very minor change to the *transformation* produces an entirely different behavior altogether, one very useful for avoiding obstacles during the docking maneuver.

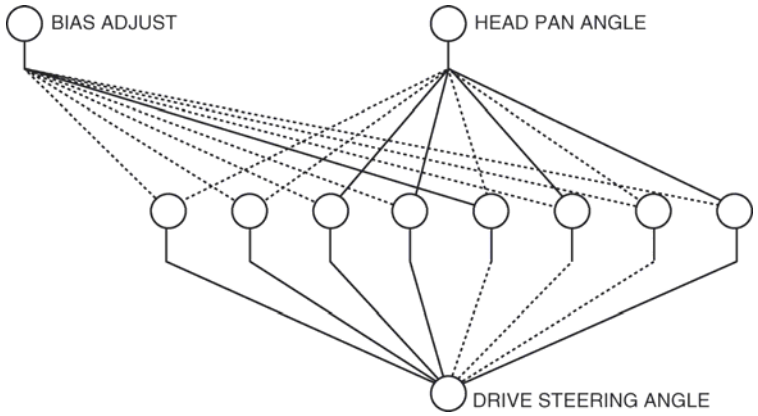


Figure 3-9. Neural-net analogy for the docking maneuver, which transforms a single sensory vector (i.e., head pan angle, or relative bearing to the beacon) to an output vector (i.e., steering angle).

3.2 Docking System

The fact that the docking procedure purposely steers the robot into contact with the recharging station presents a minor dilemma to the interrupt-driven collision-avoidance routines, which were specifically designed to reflexively avoid any such close encounters. To get around this problem, the docking routine would set register *Homing* to indicate that a docking approach was in progress. This register was always checked by the collision-avoidance interrupt routines before any avoidance response was initiated. The other necessary conditional was the robot's relative distance to the charger, as indicated by the *Range Output* status from the *Optical Board*.

If *Homing* was set and *Range Output* was high, indicating close proximity to the beacon, collision-avoidance interrupts were subsequently disabled, allowing the final stages of docking to proceed without interference. On the other hand, if *Homing* was set but *Range Output* was still low, the robot needed to execute an avoidance maneuver tailored specifically to the situation. In such cases, subroutine *Skirt* would be enabled by the interrupt routines to perform obstacle avoidance from within the docking routine itself.

Whenever *Skirt* was called, the robot would halt, then back away from the charger, simultaneously turning until the beacon was positioned almost 90 degrees right of the forward axis as shown in Figure 3-10. The platform next moved forward for a predetermined length of time, adding 07 to the beacon position as seen by the head, and using the result as a steering command for the drive motors. This very minor modification (i.e., adding an offset bias) to the *sensorimotor transformation* created an entirely new behavior. Instead of moving in a *radial* direction towards the beacon, the robot would maintain the beacon at the 90-degree position and move *tangentially* around the charger. This circumnavigation maneuver repositioned the robot to a different vantage point, roughly the same radial distance away from the beacon, but hopefully with an approach path now clear of obstacles. Subroutine *Align* then realigned the robot with the beacon, at which point normal docking resumed along a radial path.

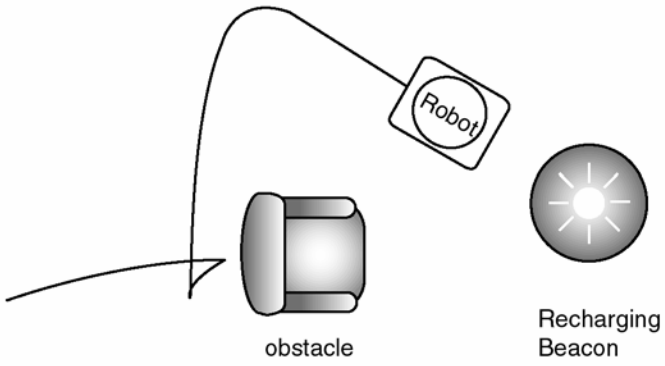


Figure 3-10. Simply adding a bias to the *sensorimotor transform* causes the robot to move tangentially around instead of radially toward the charger, thus avoiding the intervening chair.

The task of making contact with the recharger was greatly simplified by restricting the circuitry involved to the battery-voltage level (i.e., 13.5 volts), as opposed to the more dangerous 117 volts of a normal AC distribution system. This lower voltage allowed for

contact surfaces to be exposed with no electrical shock hazard. Since these surfaces were by design symmetrical with respect to the vertical pole supporting the homing beacon, they presented the same target profile, regardless of the direction of approach. The need for critical alignment to ensure effective contact was thus greatly reduced.

The metal pole supporting the optical homing beacon served as the point of contact for the GND leg, its mating surface being the front-bumper panel of the robot chassis (Figure 3-11). As discussed in the previous section, this bumper was spring-loaded to allow it to absorb impact, and to facilitate actuation of numerous micro-switches used as sensory inputs for collision detection. This inherent spring action served to keep the aluminum bumper in tight contact with the pole once the two came together, compressing the springs, whereupon the forward motion of the robot was halted. This "docked" condition was sensed by the software as a combination of tactile-switch closure in conjunction with recharging power sensed onboard.

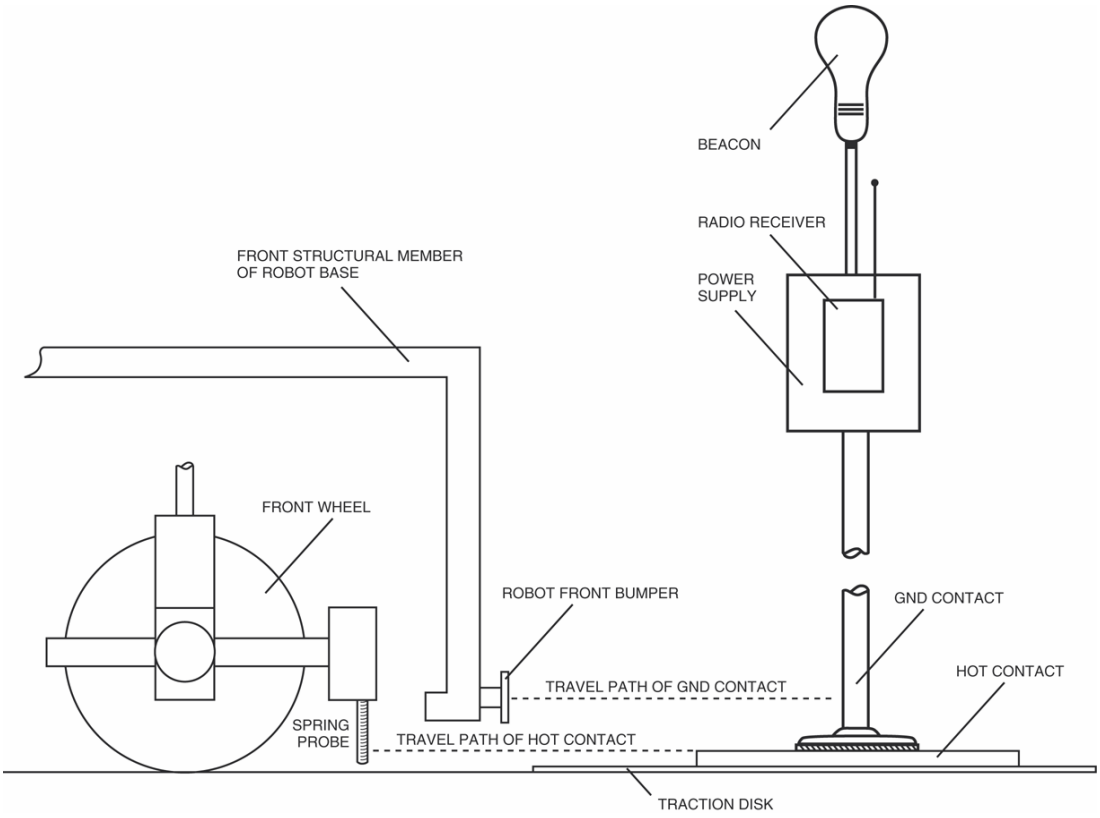


Figure 3-11. To establish an electrical connection, the spring-loaded front bumper would make contact with the metal pole, while the spring probes just forward of the drive wheel mated with the aluminum base plate. The traction disk below the base plate prevented the robot from pushing the charger across the floor.

The connection for the HOT leg of the circuit was through contact between a circular aluminum plate at the base of the beacon tower that was electrically isolated from the vertical pole (the GND connection) by a plexiglass insulator. The spring probes that mated with the circular plate extended downward from a small plastic box attached to the drive-wheel support cage. As the bumper passed over the aluminum plate moving toward the pole, the spring probes were dragged across the plate, and contact was maintained as

the platform continued toward bumper impact. The circuit was completed when the bumper touched the upright pole, at which point recharge current started flowing to the battery through an isolation diode. An electrical relay on the robot would de-energize the forward windings of the drive motors when this final connection was made, as a backup for the software, which turned off the drive motors when the recharge-probe potential went high.

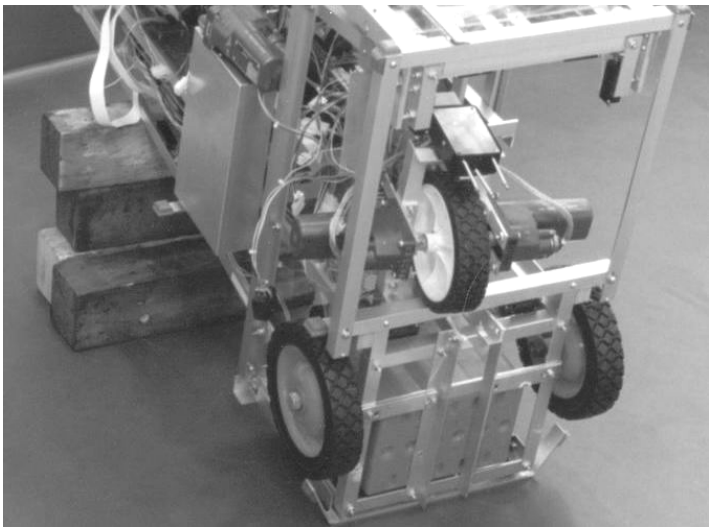


Figure 3-12. This maintenance photo taken during replacement of the steering-column bearing assembly provides a rare bottom view of the three spring probes, just forward of the front drive wheel, which formed the HOT connection.

The use of multiple spring probes (see again Figure 3-12) in the HOT leg ensured a good connection with lower current density at the mating surfaces. The software would monitor the probes as well as the battery level while the system was recharging, and if electrical contact was lost, command the necessary forward motion to re-establish the connection. In extensive testing (i.e., over 200 computer-controlled dockings), this very seldom happened, primarily because the spring-loaded bumper provided plenty of force to keep the surfaces pressed tightly together. The geometry of the configuration was such that the probes would be in contact with the plate whenever any part of the front bumper was touching the vertical pole. Since the bumper was 14 inches wide, considerable margin for error was thus allowed the tracking system that brought the robot into proper alignment. This extremely simple and inexpensive approach consistently produced final impacts well within 2 inches of centerline in repeated testing.

3.3 Recharging System

The robot's battery level was monitored by an LM339 comparator, which set a flip-flop when the voltage fell below an adjustable set-point for more than 5 seconds (see Figure 3-13). This delay was used to ensure the battery voltage was not momentarily pulled low by a stalling drive or steering motor. An interrupt was generated when the flip-flop changed state, whereupon the interrupt service routine gated out the flip-flop and selected the docking behavior (see section 4). The battery voltage was also monitored by an *LM3914 Dot Display Driver* with an associated 10-LED bargraph to give a visual indication of the battery condition. The upper LED segment in this display fed another

comparator which changed state when the battery was fully charged, as indicated by the display. This "fully charged" set-point was also adjustable as shown in Figure 3-13.

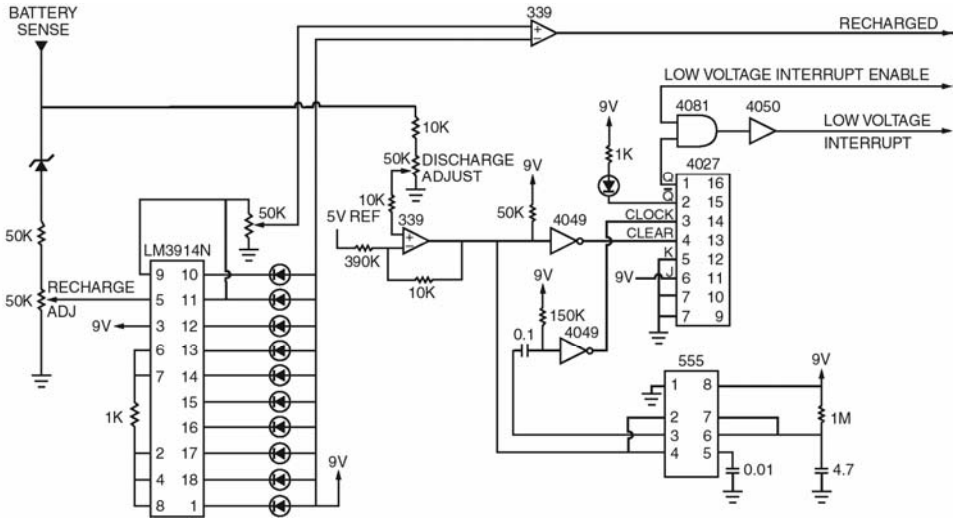


Figure 3-13. A low-battery condition detected by the LM339 comparator sets a flip-flop (4027) after a 5-second delay created by the 555 timer.

There were two power supplies associated with the recharging station itself. A relatively low-power 12-volt source remained energized at all times, supplying the RF receiver and decoder circuitry that activated the beacon. It also energized the aluminum base plate through a resistor-capacitor network to a peak potential of 16.5 volts, which acted as the "contact-sense" voltage for the pickup probe, allowing the microprocessor to know when the recharge circuit had been completed. As soon as the battery was connected as a load on this supply, this voltage level would drop to just slightly over the battery voltage (typically around 13.0 to 13.5 volts). This drop was sensed by detection circuitry on the recharging station, which in turn activated the battery charging supply. This second supply furnished the current required to recharge the battery, and was automatically shut off when the robot disconnected and the battery load was no longer sensed. The homing beacon could be deactivated via the radio link once a good electrical connection was verified, without affecting the recharger supply.

3.4 Final Thought

In closing, I would like to reiterate that the beacon-tracking and homing strategy on *ROBART I* was in no way influenced by or intentionally patterned after neural network approaches similar to the examples in this section. As stated earlier, I had no familiarity or experience with this subject whatsoever in the early eighties. I noticed the similarities between my hardwired circuitry and neural-network-based sensorimotor control while researching issues related to the human brain for *The Evolution of ROBART*. I was understandably intrigued, and in probing further, profoundly struck by the many parallels as were discussed.

There are many excellent and authoritative texts on various aspects of the brain (i.e., neurocircuitry, consciousness, cognition, agnosia), but most are written by physicians, psychologists, and/or philosophers with understandably little or no background in

robotics. I decided to treat the subject here to help bridge that gap by way of example, however simplistic and after-the-fact it may be. I firmly believe the disciplines of neuroscience and robotics share a synergistic future, and at times I see that destiny as just over the near horizon.

3.5 References

Bridgeman, Bruce, "*The Computational Brain* by P.S. Churchland and T.J. Sejnowski," review in *Psyche*, Vol. 1, No. 3, December, 1993.

Churchland, Patricia S., and Sejnowski, Terrence J., *The Computational Brain*, ISBN 0-262-03188-4, MIT Press, Cambridge, MA, 1992.

Churchland, Paul M., *The Engine of Reason, the Seat of the Soul: A Philosophical Journey into the Brain*, ISBN 0-262-03224-4, MIT Press, Cambridge, MA, 1995.

CNN, "Secret of Homing Pigeons Revealed," *Science and Space*, Reuters, CNN.com, http://www.cnn.com/2004/TECH/science/02/06/homing.pigeons.reut/, February 6, 2004.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot*, Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995.

Kreithen, M., "Orientational Strategies in Birds: A Tribute to W.T. Keeton," in *Behavioral Energetics: The Cost of Survival in Vertebrates*, Ohio State University, Columbus, OH, pp. 3-28, 1983.

Pribram, Karl, *Languages of the Brain*, ISBN 0-913-41222-8, Brandon House (Prentice Hall), Englewood Cliffs, NJ, 1971.

4
Complex Behaviors

> "All human actions have one or more of these seven causes: chance, nature, compulsion, habit, reason, passion, and desire."
>
> *Aristotle*

The *CRAWLER's* punched-card and electro-mechanical relay logic provided only two primitive functions, recharger docking and "bump-recover" exploratory motion. A more positive way of looking at it, on the other hand, would be that two meaningful autonomous behaviors had been implemented with minimal resources on a high-school science project. (I spent less than $200 total on both versions.) The integrated-circuit logic I later learned about at Georgia Tech certainly provided orders of magnitude more capability than relay technology. But a single-board computer with 64 kilobytes of address space? Clearly that represented a quantum leap in potential performance, compared to a makeshift card reader with only a handful of four-bit commands!

Excited by this prospect, I had high hopes for the behavioral sophistication of *ROBART I* relative to its *CRAWLER* ancestry. Only stand-alone software modules had been implemented up to this point, able to read inputs and manipulate outputs for such low-level support functions as tracking, steering, or docking. But so far this collection of primitives did little more than replicate the basic *CRAWLER* functionality, despite my heightened expectations. To accomplish more complex tasks, *ROBART I* needed an operating system that could sequence multiple primitives in the proper order of execution.

The actual specifics of such a process, however, were still sketchy in my mind. I wanted a simple means for human intervention, so that a particular behavior could be requested (or terminated) as needed. Some prioritization of tasks was desired, so if warranted by emergent conditions, a more appropriate routine could be substituted for the one in execution. But what about the robot's behavior when no preconceived scenarios requiring specific action were present? Constantly moving about in random fashion, for example, would be a waste of battery power, and likely also annoying. Besides, the robot's primary security mission required it for the most part to be stationary.

4.1 Operating System

After much careful consideration, the operating system I settled on addressed the following principle issues:

- Pseudo-random execution of numerous canned behaviors, so the robot's activities varied over time, as opposed to being predictably repetitive.
- Interruption of the current behavior, followed by substitution of another routine, based on the robot's assessment of prevailing conditions.
- Human intervention to accomplish similar behavior termination/substitution.
- Sequential ordering of multiple behaviors (i.e., for serial execution) to accomplish a more complex task.

What was *not addressed* during my thesis work was an on-the-fly capability to generate some new sequence of behaviors in support of the last bullet above. In a more sophisticated futuristic machine, I reasoned, this problem would likely be solved by some goal-oriented AI program that recognized needs, and then planned ways to satisfy those needs with deliberative actions. An elaborate model of the machine's perceived environment (as well its own internal state) would perhaps be constructed to assist in this regard. If the desired goal were broken down into incrementally achievable sub-goals, each designed to accomplish an intermediate objective, then potential sub-goal solutions could be first tested within the model. The train of correct choices that arrived at the required end point (i.e., achieved the overall goal) would at that point be executed, hopefully meeting with the same success as when simulated.

In a system designed around a single eight-bit microprocessor, however, all this was wishful thinking, especially 25 years ago. Furthermore, it was not the purpose of my thesis prototype to serve as a development platform for sophisticated AI research, but rather as a test platform for different type sensors, their interface circuitry, and associated software drivers. A simplistic means was therefore implemented in assembly language to provide a reasonably intelligent process of goal attainment through sequential execution of ordered primitives. I put off the more difficult problem of task decomposition, in other words, and focused instead on ordered task execution to achieve more complex behaviors.

Just as the *software* was supported by a library of generic *subroutines*, the *system* was provided a set of predetermined *behavior routines*, which could be rearranged as needed to accomplish a variety of tasks. Unfortunately, however, this rearrangement had to be done *a priori* by the programmer, versus in real time. Nevertheless, this was still a very useful feature, for it facilitated the use of generic primitives to build multiple behaviors, versus redundant code. For example, the same beacon-tracking algorithm could be called when either docking with the charger or when using the beacon as a navigational aid for entering the hallway (as discussed in the next section). In this fashion, I unknowingly began to embrace the concepts of structured programming, building reusable modules of software with defined entry and exit conditions, effectively creating a pseudo high-level language for mobile robot control (Everett, 1982).

4.1.1 Human/Robot Interface

The first operating-system feature I tackled was the need for some type of user interface. Very few mobile systems are brought online with no means of human interaction whatsoever, particularly in the initial stages of development. How much human involvement is desired and how it should be implemented must be addressed early in the design, taking into account both hardware and software issues. From a hardware perspective, there were two principle methods of human/robot interaction: (1) the *Enable/Disable Panel* on the upper front of the body, just below the head, and (2) the *User I/O Panel* on the upper left side. It is significant to note that both these interfaces were physically located on the robot, and that *ROBART I* had no RF data link (or tether) to a remote host computer or operator control unit.

Enable/Disable Panel

With regard to my previously admitted software inexperience, probably the most telling feature of all was the *Enable/Disable Panel*, made even worse by its prominent up-front location just below the head (Figure 4-1). This interface served as a hardware instantiation of enable/disable flags for various functions, and in many cases also controlled power to the associated subsystems. The very presence of this panel bears testament to the fact that I designed the hardware before accruing any significant programming skill, else I would have known better. Coming from a mostly electronics background, I naturally fell back on a hardware solution, not realizing the same thing could be accomplished by setting and clearing software flags in embedded conditionals. I learned the hard way that it is a lot easier to change your code (even assembly language!) than to rewire dedicated circuitry.

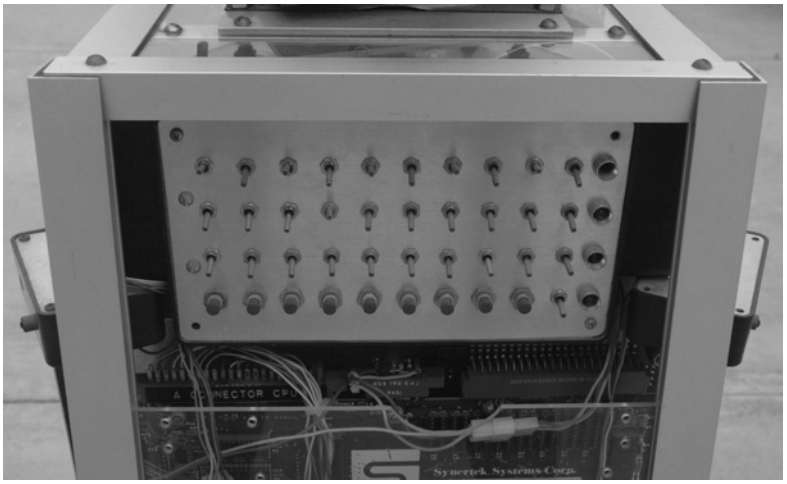


Figure 4-1. The hardware-based *Enable/Disable Panel*, which provided both development and run-time flexibility, was a reflection of my software inexperience.

User I/O Panel

The *Enable/Disable Panel* was largely a set-and-forget configuration used mainly during development and testing. The primary method of operator interface during run-time was the *User I/O Panel* shown in Figure 4-2, which addressed several things:

- A means for determining if the operator desired to take (or relinquish) control.
- Passing of control at the appropriate point in the program flow, so as not to disrupt events that should be terminated before others were initiated.
- Determining which routine the operator wanted to perform or request.
- Inputting any parameters needed to execute or clarify the chosen routine.

Figure 4-2. The *User I/O Panel* enabled some operator interaction with the behavior of *ROBART I*, which lacked any remote *operator control unit* (OCU) or *graphical user interface* (GUI) as we know it today.

Subroutine *Control* would perform all the above functions, interacting with the operator through voice prompts, which made the otherwise non-intuitive process fairly easy to follow. *Control* checked the four input switches on this panel (see again Figure 4-2), and if any were set, would disable interrupts, stop the drive wheel, and store the old drive command in register *Dri.com*. The operator would be instructed to set the external switches specifying the four-bit *Service Select* code, and then press the *ENTER* button. This four-bit code determined which service the operator was requesting, and was stored in register *Ser.cd*. The software then requested the "Control Entry," which was input in a similar fashion and stored in register *Con.cd*. Subroutine *Op/exec* (operation execute) would then perform the required service based on the contents of register *Ser.cd*, subject to the constraints represented by the parameter *Con.cd*.

This binary toggle-switch interface was admittedly very crude, but it was seldom ever used under normal conditions, as *ROBART I* was a fully autonomous system. The primary purpose of the *User I/O Panel* was to force a particular behavior during demonstrations or while troubleshooting. There were 15 *Service Routines* the operator could request in this manner, with the special code of zero used to return control back to the onboard computer. Subroutine *Pl.dri* (pull drive) would be called, which fetched the old drive command from register *Dr.com*, IRQ interrupts were re-enabled, and the software resumed where it had left off before shifting to operator control.

4.1.2 Behavior Sequencing

The operating system was structured around one main loop that controlled branching to the various behavior routines, each of which had its own exit requirements. Individual subsystems were energized and tested when the system was first brought online (see

Figure 4-3). Verbal instructions were issued to correct any detected discrepancies, such as *Enable/Disable* switches in the wrong position, and any significant errors would initiate shutdown procedures. Following successful startup and diagnostics, the first routine was randomly chosen from a possible range of 0 to 15. (The available options here were further decreased to include only routines 0 through 7 if the *Drive Power* switch was in the *Off* position, as behavior routines 8 through 15 involved vehicle motion.)

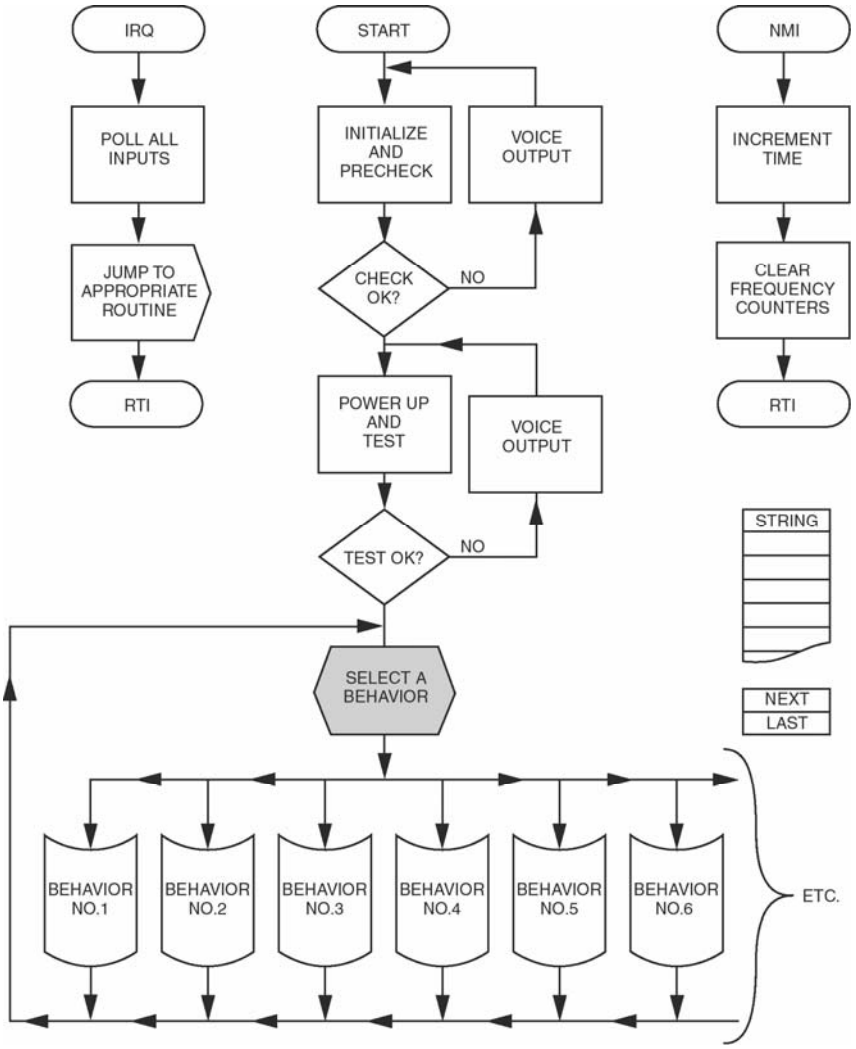


Figure 4-3. Simplified block diagram of the software operating system on *ROBART I*.

The behavior-selection process highlighted in Figure 4-3 above chose a routine for execution in accordance with the following priorities, as further amplified in Figure 4-4:

- See if a follow-on behavior had been specified in *Next*.
- Else see if a sequence of behaviors had been specified in *String*.
- Else see if the security *Arm/Disarm* switch was set to *Arm*.
- Else randomly select a behavior from a pre-designated set.

The same behavior would not be selected twice in succession, and certain sensory inputs could alter the probability of a routine being chosen. While the individual behaviors were not recursive, each could call within itself any of the others. The system kept track of the last routine executed, any routine interrupted by another of higher priority, and the next one slated for execution. If the next routine was not specified, then one was chosen at random, but within the constraints imposed by internal and external conditions.

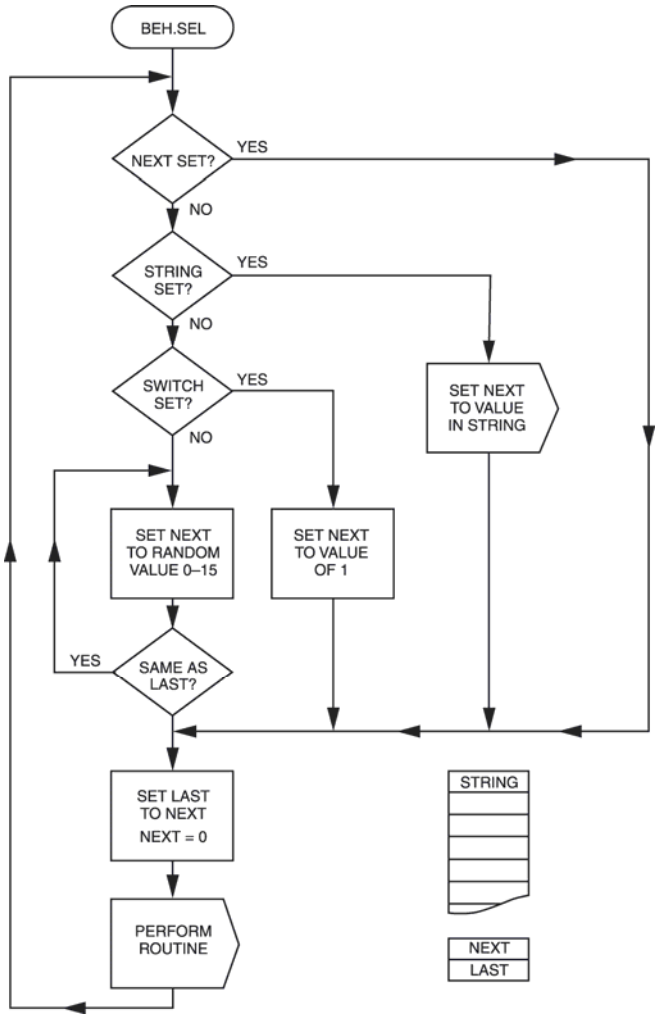


Figure 4-4. Expanded block diagram of the behavior-selection process (see Figure 4-3) that enabled sequential execution of primitives, with some basic ability to override lower priority tasks as needed.

Specifying the Next Routine

In the selection process outlined in Figure 4-4 above, register *Next* was first checked to see if a specific behavior had been specified by the operator or some other source. For example, *Operator Service Number 6* could be used to manually set the contents of register *Next*, if so desired, and thus specify the subsequent routine within the subset of 1 to 15. Alternatively, *Next* could be set by the onboard software under certain pre-specified conditions, as discussed later.

Specifying a Behavior Sequence

If *Next* were clear, then the one-dimensional array *String* was examined to see if a series of behavior primitives had been requested. If register *String* was non-zero, its value would become the next routine, and each array value would shift up one location in the 16 consecutive memory bytes beginning with *String*. (In other words, the software manipulated this array of registers beginning with *String* as a stack, to where *String* would take on the value of register *String + 1*, and *String + 1* the value of *String + 2*, etc.) This stack was limited in that if more than 15 routines were saved, the original numbers fell off the end and were lost. The 16^{th} array value was always maintained at zero to flag the string end. Each time a value was taken from *String*, the remaining array entries each moved up a space to replace it, until *String* eventually took on the delimiter (i.e., end-of-array) value of zero.

If *String* was clear, then the *Hostile/Friendly* switch on the *Enable/Disable* panel would be checked. If this switch was in the up position, it would force the selection of Behavior Routine Number 1, which monitored for intrusion. (This feature provided a one-step shortcut for the most frequently requested behavior.) If none of the above three sources specified the next routine, however, a random choice was made, compared with register *Last* to ensure no back-to-back duplication, and then performed. Although extremely limited in scope, this necessarily simplistic means for sequencing behavior primitives was nonetheless effective, foreshadowing later use of a more sophisticated script-file interpreter on *ROBART II* and the *MDARS* robots.

Behavior Termination/Substitution

The cancellation of an active routine before its normal completion point was another desired feature, whether internally decided by the software or externally requested by an operator. This function was implemented through subroutine *Termin*, which set register *Exit* if the ENTRY button on the *User I/O Panel* had been pressed. *Exit* then served as a global flag to terminate any looping behavior routine in execution. Whenever a critical routine was cancelled to allow a higher priority routine to take place, however, provision had to be made for resumption of the original routine upon completion of the substitution. This capability was accomplished by subroutine *Ph.strg* saving the interrupted routine number in *String*. Just as the register contents moved up as routine numbers were taken from *String*, they moved down one location each time a number was saved.

Additional Behavior Routines

While the random selection process was arbitrarily limited to routines 0 through 15, the total number of behaviors was constrained only by available memory space. The randomly selected routines were designed to produce fill-in behavior during periods where specific activities, such as surveillance or recharging, were not needed. Those routines above 15 dealt with situations requiring definitive action on the part of the robot, and could be neither randomly chosen nor selected by operator control.

4.2 Application Behaviors

The first complex behavior implemented on *ROBART I* was automatic recharging, but that was a self-serving sustainment behavior as opposed to a useful application. From a value-added perspective, there is something decidedly lacking in a system that exists only to feed itself, although I must admit I have run across a few humans who do fit that description. (All kidding aside, there are many natural organisms that do just that, live to eat, for their sole purpose is to serve as nourishment for more sophisticated life-forms higher up in the food chain.) For an autonomous robot to be of any significant use, it likewise must have a meaningful reason for being. I selected the *security* application to formally address in my thesis work, but also experimented somewhat with the secondary roles of *education* and *entertainment*.

4.2.1 Security Applications

It seemed fairly obvious that the role of a security robot should be to alert the building occupants to any abnormal conditions, such as fire, smoke, toxic gas, flooding, earthquakes, approaching storms, and potentially even intrusion. Fire and smoke detectors were readily obtainable, and a little sleuthing turned up an inexpensive toxic gas sensor made by Figaro (Everett, 1982). I adapted a simple continuity circuit for a basement flooding monitor by dragging a pair of linked-ball keychains across the floor beneath the robot's rear wheels. The measured impedance between these two sense elements would drop substantially if they encountered a wet floor surface.

The earthquake sensor (Figure 4-5) required a bit more innovation, in that nothing short of laboratory instrumentation seemed available at the time, and that option was outside my collective volume, power, and cost budgets. Its inclusion was inspired by our recent relocation to Monterey, which of course had prompted all our east-coast friends and relatives to offer up an assortment of dire predictions as to when California was going to slide off into the ocean. (In retrospect, having experienced a fair share of both, I will take an earthquake over a hurricane any time.)

Figure 4-5. The earthquake detector (center) consisted of a free-floating rod inside a vertical brass tube, mechanically coupled at the base to a crystal microphone element.

To address this deficiency, I customized a low-frequency vibration sensor by inserting a 12-inch length of coat-hanger wire inside a ¼-inch-diameter brass tube, vertically orientated as shown in Figure 4-5, and mechanically coupled to a modified crystal microphone. If the robot chassis started to sway even a little bit, the coat-hanger would rattle back and forth inside the tube, directly stimulating the crystal element. The amplified signal associated with this disturbance was appropriately filtered and fed to a threshold comparator, which then generated an alarm interrupt. The comparator output was of course disabled whenever the robot was in motion.

The thunderstorm detector was a similar hack, comprised of an AM radio tuned to an unused frequency in the broadcast band, essentially listening for static caused by lightning discharges. The rectified audio output was applied to a small capacitor on the input of another threshold comparator, which besides generating an alarm interrupt for the *SYM* would also activate a 24-hour weather radio. The associated electronics were mounted in the black plastic enclosure atop the head as shown in Figure 4-6. But while it rained quite a bit in Monterey, we did not get that much thunder and lightning, so this novel feature never saw a lot of practical use.

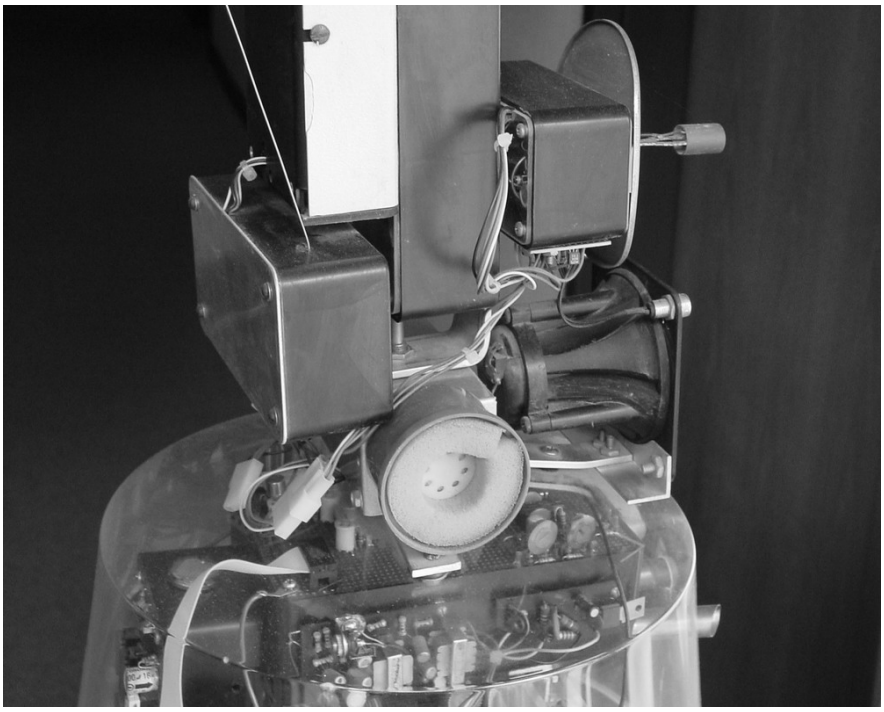


Figure 4-6. The thunderstorm detector was housed in the black plastic enclosure at left center in the above photograph, with the whip antenna extending above and to the right. The right-hand acoustical-detection microphone can be seen at lower center, surrounded in foam padding.

We also did not get a lot of intruders in the Navy housing where I lived as a student, but that did not stop me from placing a major emphasis on reliable intruder detection.

Intrusion Detection

The first intrusion-detection modality to become operational was discriminatory hearing, with a bandpass filter to differentiate those sounds likely associated with forced

entry, such as breaking glass, sawing, or filing, etc. More common household noises, (i.e., a furnace, air conditioner, or refrigerator) were greatly attenuated by the filter, and therefore did not reach the threshold required to trigger detection. I stumbled upon this capability as an offshoot of my earthquake detector design, in that the microphone element used to sense vibration would often be falsely activated by ambient noise. That problem had been easily resolved by removing the acoustical diaphragm and coupling the brass sensor tube directly to the crystal element. But for this application, where the intent was to detect noise, I employed two highly sensitive omnidirectional microphones that served as the robot's ears. Each microphone was mounted inside the plastic lid from a can of shaving cream, effectively isolated from parasitic vibration by a surrounding cushion of foam padding (see Figure 4-6).

The forward-looking collision-avoidance sonar could also be used in an intrusion-detection mode when the platform was not moving, simply by recording the range to the nearest target. If a person subsequently walked through the sonar field of view and decreased this range, the software would respond accordingly. Two fundamental problems severely limited the practicality of this approach, however; there was only one sonar sensor with fairly limited range (i.e., typically 4 feet or less to a human target), and its performance was rather finicky even under ideal conditions, as previously discussed. Consequently, I abandoned this strategy on *ROBART I* and took it up several years later with *ROBART II*, which had a ring of 24 sonars (Figure 4-7) for 360-degree coverage out to an effective human-detection range of about 23 feet (Everett, 1985; Everett & Gilbreath, 1989).

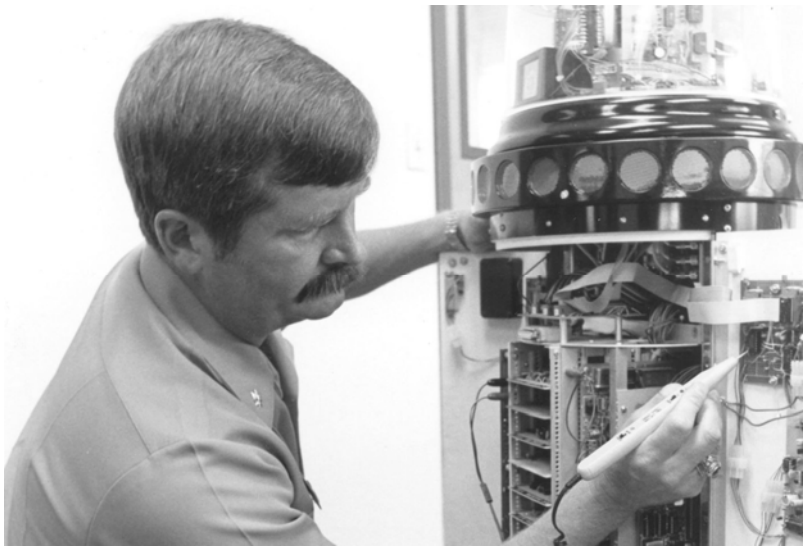


Figure 4-7. *ROBART II* (1982-1992) employed a 360-degree ring of 24 Polaroid® sonar transducers (situated just below the head) for use in navigational referencing and intruder detection.

Adding an optical motion-detection system provided even more capability. Three National Semiconductor® *D-1072* integrated circuits were used to detect changes in ambient light level (Figure 4-8). This special-purpose chip, which incorporated a built-in photodiode and plastic lens, was completely passive and sensitive over the dynamic range of 0.1 to 100 candlepower (Weiss, 1979; Gontowski, 1983). The cone-shaped detection

zone created by the lens covered a 2-foot circle at approximately 8 feet, and considerable range was possible, depending on background lighting conditions. After a brief settling period upon power-up, the circuit adjusted itself to ambient conditions, and any subsequent deviation from that setpoint would result in an alarm output. These optical detectors were effective only when the vehicle was stationary, and therefore intentionally disabled when the platform was in motion.

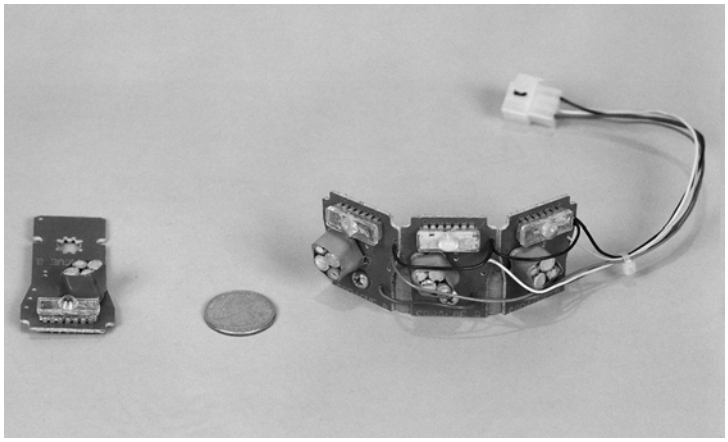


Figure 4-8. Three National Semiconductor® *D-1072* optical motion detectors, configured in a diverging array (right), responded to changes in ambient light level.

The low cost and directional nature of the device allowed for several to be used in an array to establish unique detection zones that helped establish the relative position of the suspected security violation. Three sensors were situated with their respective fields of view covering slightly overlapping regions: left, right, and straight ahead with respect to the robot's head position. Unfortunately, the *D-1072* suffered from three drawbacks that limited its utility and contributed to eventual discontinuation: 1) the power consumption of the device was fairly significant, 2) it required some degree of ambient background illumination, and 3) it was rather susceptible to nuisance alarms (i.e., false triggering by naturally occurring stimuli).

By far the most sophisticated sensor for intrusion detection was a passive-infrared (PIR) motion detector sensitive to body heat (Figure 4-9). This device was a recently introduced off-the-shelf unit manufactured by Colorado Electro-Optics, and quite an innovative concept at the time. The sensor responded only to discrete thermal sources that were actually moving, with a cone-shaped detection zone that fanned out to a width of 20 feet at its advertised 50-foot maximum range. To the best of my knowledge, *ROBART I* was the first mobile robot ever equipped with such a sensor (Everett, 1982). Similar implementations were later reported by Quick (1984) and Cima (1984).

In conventional security applications, these devices are mounted such that the sense element "stares" at a stable thermal field-of-view, and responds only when a moving entity disturbs the magnitude and distribution of incident photons (Cima, 1984). The principle of operation as a motion detector is similar to the *D-1072* optical sensor described above, except a different range of wavelengths (7 to 16 micrometers) in the energy spectrum is sensed. The *emissivity* of human skin is very close to unity (0.98) and the same for all races (Cima, 1984). A typical human gives off somewhere between 80

and 100 watts of radiant energy with a peak wavelength around 10 micrometers (Cima, 1990), thus producing a distinctive thermal signature.

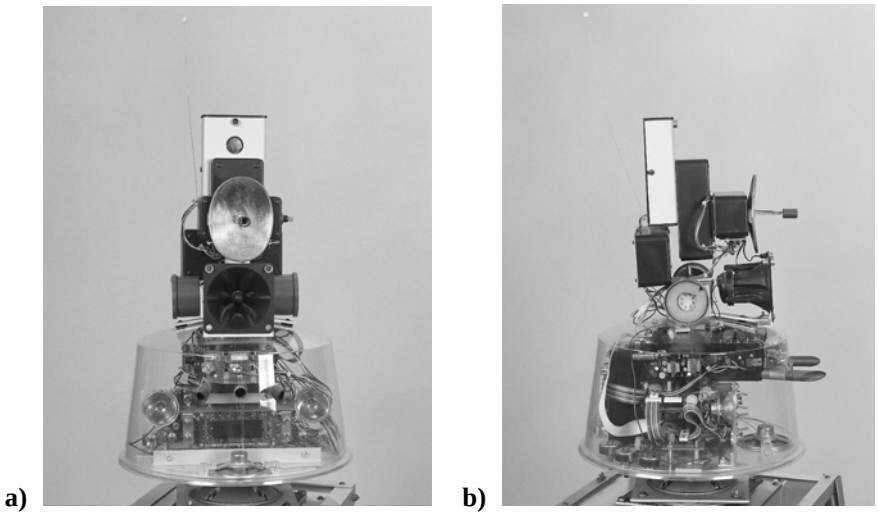


Figure 4-9. a) Front and **b)** side views of the Colorado Electro-Optics passive-infrared motion detector (white enclosure at top), which proved extremely effective as a primary security sensor.

Most commercially available systems incorporate opposed-output, dual-element detectors that provide common-mode rejection of stationary thermal disturbances (Figure 4-10a). When a human target moves left to right through the sensor's field of view, however, the focused concentration of photons in the image plane travels right to left across the two detector elements. The lens geometry is such that the incident radiation falls almost exclusively at first on the right-hand detector, reaches a balance between the two as the intruder crosses the optical axis, and then shifts with continued motion to where the left sense element dominates. The resulting output signal is plotted as a function of time in Figure 4-10b. If the "intruder" stops moving at any point while still in view, the detector will settle out to equilibrium with no appreciable output signal.

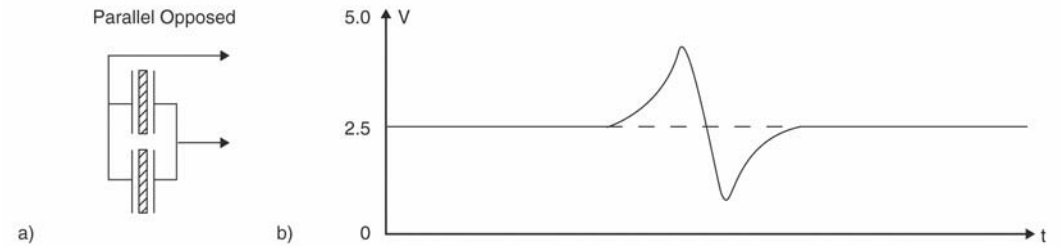


Figure 4-10. A typical output signal of a dual-element PIR detector showing the characteristic rise and fall signature relative to the 2.5-volt equilibrium (adapted from Jones & Flynn, 1993).

Motion Detection on the Move

The proverbial Achille's heel of this intrusion-detection sensor suite was that the more reliable sensing modalities exploited human mobility as a key means of minimizing nuisance alarms, and this meant they were limited to static monitoring when the robot was stopped. The development of cost-effective detection methods that could operate

from a moving platform was then a significant area of interest (Everett, 1982), and still is some 25 years later.

A passive infrared sensor, in general, is most sensitive to cross-walk and least responsive to distant objects moving radially towards (or away from) the unit. By exploiting this characteristic, the Colorado Electro-Optics sensor was found to be stable enough under certain conditions to operate when in motion. If the robot's head was positioned slightly off-center as the platform moved slowly forward, the presence of a stationary intruder on that side would be detected due to its relative motion with respect to the moving vehicle. Alternatively, a similar result could be achieved by slowly turning the head while stopped, effectively scanning a circle of 50-foot radius. This capability allowed the robot to stop in a doorway and from that vantage point scan the room interior. An intruder present would generally be detected, even if able to remain motionless during the sweep.

The effectiveness of this infrared sensor in non-stationary applications varied somewhat with background temperature, however. Under reasonable cool ambient conditions (i.e., 65 to 70 degrees Fahrenheit), the human body, an excellent emitter of infrared energy, will produce a striking temperature gradient. If vehicle velocity and/or the speed of head rotation were kept relatively low, I found that false triggering of the sensor could be minimized. Since the robot responded to a moving detection by stopping and bringing other sensors to bear for confirming indications, one or two false alarms in this mode created no real problem. After a brief wait, the robot would conclude there was nothing there and resume patrol.

The fundamental problem with this approach was the presence of thermal gradients other than those due to human presence. Warm bodies are normally distinguished from these alternative sources by virtue of their mobility, which after all is the precisely the feature exploited by a fixed-mounted PIR motion detector. If the detector itself is translated or rotated to create the necessary relative motion, however, this distinguishing characteristic is lost. A stationary human target will appear no different than any other static thermal source, such as a lamp or heater. As a consequence, this method of intrusion detection on the move is strictly limited to operational scenarios free of competing thermal gradients, such as for example an unheated warehouse (unless some temporal assessment of change is used to ascertain motion).

The *Lifescan FLIR-6* was a commercial product based on this "scanning-detector" concept, introduced sometime in the late eighties as a hand-held human-presence sensor for law-enforcement applications. This battery-powered unit was about 8.5 inches long and weighed just slightly over a pound, roughly the size and weight of a typical flashlight (Figure 4-11). In addition to a red output LED, the device incorporated a small internal vibrator to alert the user to an alarm condition. We purchased a couple of these systems for evaluation as possible sensors on *ROBART II*, but unfortunately their performance did not appear too repeatable for reasons presented above. To the best of my knowledge, they are no longer available.

Figure 4-11. The *Lifescan FLIR-6*, a handheld PIR human-presence sensor intended for law-enforcement use, was slowly scanned back and forth by hand to detect static targets.

Herb Viggh and Anita Flynn (1988) describe a rotating pyroelectric detector expressly developed to support a human-following behavior on the MIT mobile robot *Seymour*. Their innovative system was based on two Eltec *Model 442* PIRs (Eltec, 1991), which operated in a trans-impedance current mode more suited to a continuously scanned implementation. A *synthetic field of view* defined by the leading-edge detections associated with each of two sensors was created by rotating them as shown in Figure 4-12. The elapsed time interval between individual sensor detections was a function of the separation distance between the robot and the perceived thermal source, and thus indicative of range to a suspected human target (Everett, 1995).

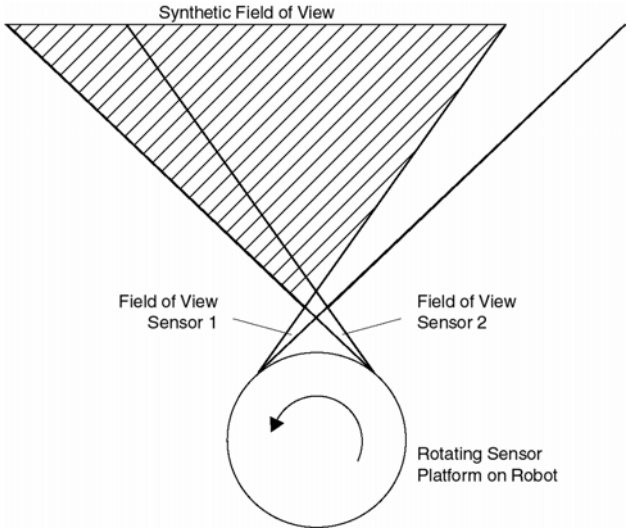


Figure 4-12. A *synthetic field of view* (shaded region) created by two rotating PIR sensors yields a varying time delay between leading-edge detection of a stationary person that is proportional to range (adapted from Viggh & Flynn, 1988).

A very successful scanning-PIR configuration for static surveillance was later developed by Cybermotion, Inc.® for the *Security Patrol Instrumentation (SPI)* module (Figure 4-13) on their *SR2* robot (Holland, 1993). The *SPI* incorporated a scanning rotor

driven by a small DC motor at 60 rpm, with slip-ring connections for four sensor modules spaced 90 degrees apart:

- Two vertical PIR arrays
- Continuous-wave K-band microwave motion detector
- Hamamatsu® ultraviolet flame detector

Each passive-infrared array consisted of four Eltec *Model 442* sensors stacked to achieve an instantaneous field of view of 5.6 degrees horizontal and 31.6 degrees vertical.

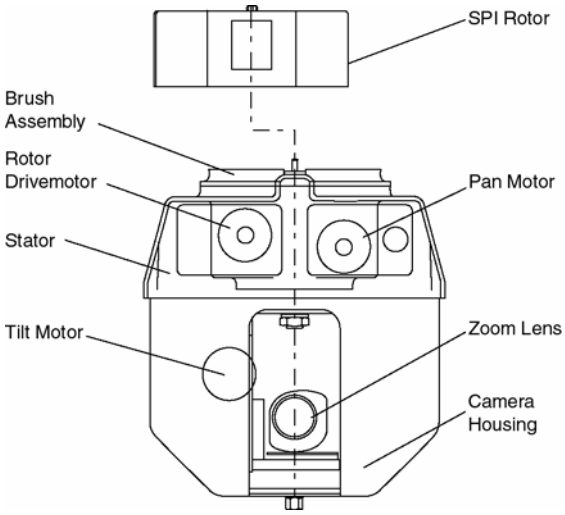


Figure 4-13. The Cybermotion® *Security Patrol Instrumentation (SPI)* module, with an integrated surveillance camera pan-and-tilt, incorporated two rotating PIR arrays (adapted from Everett, 1995).

Nuisance Alarms

One of the biggest problems encountered with conventional automated security equipment is the occurrence of nuisance alarms. Since the optical motion detectors were sensitive enough to respond to changes in ambient light due to a person walking through their field of view, they were by nature highly susceptible to spurious activation. When incorporated in a fixed alarm system, these sensors would be optimally situated to prevent exposure to changing light conditions from natural sources. When mounted on a mobile platform, however, they often may end up pointed directly at a window. In this non-optimal orientation, it is quite possible the detector could be tripped by the headlights of a passing automobile, or flashes of lightning, for example.

Similarly, specular reflection, air currents, and ambiguous effective beamwidth can potentially interfere with the operation of ultrasonic transducers (Everett, 1995). Acoustical sensors might inadvertently respond to thunder, or noise generated in an adjoining or overhead apartment. The system must be able to distinguish these common occurrences from a real intrusion scenario. If the response to an initial disturbance is used to direct the attention of all sensors to a possible alarm condition (perhaps even through relocation to a better vantage point), then all modalities could be polled for secondary indications confirming the likelihood of an actual intruder. This concept of cross-correlation to minimize nuisance alarms became the basis for the intelligent

security-assessment algorithms later pursued on *ROBART II* (Everett, 1985), and eventually awarded U.S. Patent Number 4,857,912.

4.2.2 Educational Applications

The second application area I considered was that of a robotic tutor. In early 1982, I wrote a Basic program that taught my kids math, printing out simple problems on the *KTM's* monitor screen while posing the same questions through speech synthesis. They had to type in the correct responses, which were instantly graded as correct or incorrect. The program ramped up the complexity over time as a function of observed success. I also played around briefly with a spelling tutorial, but was somewhat hampered by the *DigiTalker's* fixed vocabulary set, which limited the number of candidate words the robot could speak. Today we can do so much more with images, video clips, 3-D graphics, unlimited text-to-speech algorithms, and ever improving voice recognition.

Indeed, there are already many impressive computer-based interactive learning programs offered by a variety of vendors. The robot, however, adds another important dimension, assuming the role of mentor, talking to the child and waiting for his or her screen or voice responses. If the correct answer is forthcoming, the robot responds with positive reinforcement in the form of synthesized speech. If he or she misses one, the helpful robot can diverge to more detailed explanatory material, verbally complementing the screen depictions as needed for emphasis. With a *Wi-Fi*® or *Blue-Tooth*® interface, they can collectively surf the web for answers, illustrative examples, or more detailed explanations, enabling the robotic tutor to teach all kinds of things, perhaps even music and voice lessons. The possibilities here are basically endless!

A rather useful functionality here, especially for younger children, would be reading books aloud. In early 2005 I saw a preliminary demonstration by Evolution Robotics, Inc.® (Pasadena, CA) that involved some position-invariant image processing software running on a laptop that could read the text right off the pages of a children's book. Such a capability could be applied not only to the task of teaching preschoolers to read, but also for entertainment purposes, perhaps even for disabled individuals.

4.3 References

Brooks, Rodney A., *Flesh and Machines: How Robots Will Change Us*, ISBN 0-375-42079-7, Pantheon Books, Random House, New York, NY, 2002.

Cima, D., "Using Lithium Tantalate Pyroelectric Detectors in Robotics Applications," *Eltecdata #112*, Eltec Instruments, Inc., Daytona Beach, FL, 1984.

Cima, D., "Using Optical Radiation for Security," *Eltecdata #124*, Eltec Instruments, Inc., Daytona Beach, FL, December, 1990.

Eltec, "Model 442 IR-Eye Integrated Sensor," Preliminary Product Literature, Eltec Instruments, Inc., Daytona Beach, FL, April, 1991.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot,* Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982.

Everett, H.R., "A Second Generation Autonomous Sentry Robot," *Robotics Age*, pp. 29-32, April, 1985.

Everett, H.R., and Gilbreath, G.A., *ROBART II: A Robotic Security Testbed*, NOSC Technical Document 1450, Naval Ocean Systems Center*, San Diego, CA, January, 1989.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995.

Gontowski, W., "Build a Motion Detector Alarm," *Electronic Experimenter's Handbook*, pp. 56-64, 1983.

Holland, J.M., "An Army of Robots Roams the Night," International Robot and Vision Automation Show and Conference, Detroit, MI, pp. 17.1-17.12, April, 1993.

Jones, J.L., and Flynn, A.M., *Mobile Robots: Inspiration to Implementation*, AK Peters, Ltd., Wellesley, MA, p. 113, 1993.

McKibillo, "Welcome to the Robot Expo," *Popular Science*, pp. 58-61, November, 2005.

Quick, C., "Animate vs. Inanimate," *Robotics Age*, Vol. 6, No. 9, August, 1984.

Viggh, H.E.M., and Flynn, A.M., "Infrared People Sensors for Mobile Robots," SPIE, Vol. 1007, Mobile Robots III, Cambridge, MA, pp. 391-398, November, 1988.

Weiss, M., "Protect your Valuables: Light-Sensitive Security Alert," *Radio Electronics*, April, 1979.

* Now Space and Naval Warfare Systems Center San Diego.

5
Early Navigation

"You can't tell which way the train
went by looking at the tracks."

There is a definitive milestone in the evolutionary development of an autonomous mobile robot that every would-be creator must eventually face: the basic platform has been fabricated, wired up, and is ready for programming. Now what? How does one go about transforming this creation with so much perceived potential into an intelligent entity that the very word "robot" seems to suggest? In other words, how do we successfully achieve the autonomous mobility functions that inherently define a mobile robot? It does not appear all that difficult on the surface, thanks in part to Hollywood optimism, until you actually attempt to do it.

Indeed, in the U.S. alone, there have been literally thousands of government-funded programs in robotics over the past several decades, often involving teams of highly skilled and experienced researchers with budgets measured in multiple-millions of dollars. Yet despite such efforts, there are even today no fielded military robotic systems that can self-navigate, although as of 2004 there were over 100 of teleoperated bomb-disposal robots (Update, 2004). Similarly, only a handful of commercial applications have emerged, limited in most cases to carefully focused and well-bounded scenarios. In retrospect, one of the biggest stumbling blocks to success has been achieving a practical and reliable capability for autonomous navigation. An explosive-ordnance-disposal technician is more than willing to manually operate a system that can remotely neutralize a bomb, however long it takes, because any alternative puts human lives seriously at risk. But subscribing interest in a teleoperated floor-mopper is not similarly motivated, unless involving some hazardous clean-up operation.

Given the magnitude of this technical challenge, how then was a fledgling robotics enthusiast supposed to overcome such a daunting undertaking with the technology of the early eighties? Most of us did not have access to massively parallel computer architectures and exotic laser-based sensors. Yet this very deficiency in available resources is at times a most valuable asset, for necessity is the mother of invention, as the cliché goes. A persistent hacker who does not know a particular problem is supposed to be insurmountable will sometimes find a very practical way to get the job done, simply

because he or she lacks the funds to explore some of the more expensive alternatives. This is especially true if we bound the problem and do not pursue a 100-percent solution addressing the full spectrum of assorted difficulties. In keeping with this spirit, my early work with *ROBART I* followed a simplistic evolutionary approach for incrementally bridging this technological hurdle and attaining true autonomy in an ordinary home environment.

5.1 Behavior-Based Navigation

Simply put, a successful mobile robot must be able to get from point A to point B, and do so without running into anything. Autonomous mobility thus requires two distinctly separate but closely coupled behaviors: *navigation* (i.e., from A to B) and *collision avoidance*. All my efforts to this point had focused exclusively on the latter, with minimal provision for the former. The 10 near-infrared proximity detectors described in section 2 were sufficient to move safely about an interior structure, with the extensive array of tactile sensors serving as backup. The *Wander* software simply commanded the robot to move forward as long as no obstructions were detected, and the interrupt-driven collision-avoidance routines took over to avoid obstacles as they came into view. The ensuing motion would randomly explore the surrounding environment, but in a very unpredictable and inefficient fashion.

ROBART I needed a much better navigational strategy to be considered a convincing feasibility demonstration of an autonomous security robot. As it zig-zagged back and forth down the hallway, deflecting from first one wall and then the other, its meandering trajectory resembled that of a drunken sailor negotiating a narrow pier on his way back to the ship. Or in engineering as opposed to nautical terms, the system was under-damped and therefore overshooting. I needed a way to control the *degree* of reactive avoidance, and straighten the zig-zag out into a straight line that followed the wall. If the robot could follow a wall, it could negotiate a hallway, and hallways, I reasoned, were the backbone of indoor navigation.

5.1.1 Wall Following

I had planned on using two more *LM1812* sonars (i.e., one on each side) to support this *wall-following* behavior, but the performance just had not measured up, so I discarded that option. (The empty aluminum enclosures intended for this circuitry can still be seen just above and slightly forward of the rear wheels in Figure 5-1). But maintaining lateral offset with a *reflective proximity sensor* as opposed to a *time-of-flight ranging* device was somewhat problematic, unless I could figure out a way to exploit the sensing geometry. There was already a proximity sensor to *start* the wall-avoidance action, so why not simply add a second unit to *terminate* the procedure when the robot's heading was parallel to the wall? This closed-loop control of the turn function should keep the system from overshooting.

Fortuitously, a second proximity sensor had already been installed, logically ORed to the first to expand the vertical coverage (Figure 5-1). All I had to do, hardware-wise, was rewire it to be an independent input and decrease its fan-out angle with respect to the longitudinal axis, thereby establishing a different set point for detection. I could then

rewrite the software to ignore this sensor as an *initiator* of wall-avoidance, but poll it to determine when to end any such maneuver that had already triggered. Thinking I might be onto something, I skipped my afternoon classes to make the necessary changes, then watched with mild disappointment as the newly modified system went through its paces. Performance was decidedly better, but the robot still had a tendency to drift away and eventually cross over to the other side of the hallway, which was decidedly inelegant. I wanted the algorithm to pick a wall and stay with it.

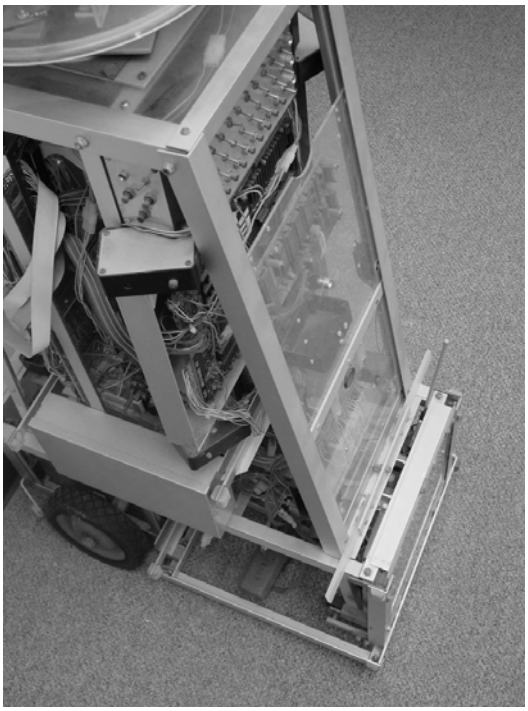


Figure 5-1. Two side-facing, near-infrared proximity sensors, aligned 60 and 40 degrees to the longitudinal axis, can be seen mounted above and below the *SYM-1* computer enclosure in this 2005 photo. The aluminum housing just above the rear wheel was intended for an *LM1812* side sonar.

Much like our housecat Tuxedo did, I suddenly realized! Tuxedo (Figure 5-2) was a very affectionate people-friendly cat that, rather than follow, liked to trot on ahead in anticipation of where you might be going. In 6 months of trailing Tuxedo to various destinations, I eventually noticed he never seemed to traverse the center of the hallway. Instead, the adventuresome cat preferred to stay very close to one wall or the other, usually deflecting his tail in the appropriate direction to provide tactile feedback. I had been so focused on this "active-antenna" tail action that I hadn't picked up on the wall hugging.[3] While Tuxedo's instinctive behavior most likely evolved as a means of survival (i.e., avoid open areas devoid of cover) rather than for navigation, it was still inspirational. Instead of a stand-off *wall-following* capability, maybe what *ROBART* needed was a cat-like *wall-hugging* behavior.

[3] On page 71 of *Sensors for Mobile Robots* (Everett, 1995), I describe a couple of *active-antenna* tactile sensors reported by Ferrel (1994) and Kaneko (1994).

5.1.2 Wall Hugging

In re-examining the specifics of the avoidance software, I remembered having inserted a temporal constraint as part of the original exit conditional. In other words, once the sensor cleared (i.e., no longer saw the wall), there was a short delay before resuming straight-line motion, so the avoidance behavior would have some hysteresis and not oscillate around the set point. But for *wall hugging*, I now theorized, which was to be a navigation as opposed to avoidance behavior, perhaps we *wanted* to oscillate around the set point, and *not* aggressively avoid. I tested this hypothesis by removing the delay and sending the robot yet again down its well-trodden path. The results were most gratifying: it still zig-zagged slightly, but the drunken sailor was beginning to sober up.

Figure 5-2. Tuxedo the Navigator, in Monterey, CA, circa 1981.

In thinking through this situation a bit further, it eventually dawned on me that a complete revision of basic philosophy was required to ensure this new wall-hugging behavior performed as desired. I had been trying to achieve a rather precise state of controlled avoidance, but without the necessary sensor resolution and repeatability to guarantee stability. What the algorithm really needed was an overriding tendency to constantly seek out the wall, with a secondary avoidance feature to prevent actual contact. In other words, incorporate two opposing behaviors that balanced each other out: one that tried to approach walls, and another that tried to avoid obstacles, including walls. If I could impose some type of steering bias that would cause the robot to drift in the direction of the wall, the standard avoidance routine would eventually deflect it away, whereupon it would slowly drift back again. This *persistent convergence* should cause the robot to aggressively hug the wall in a piecewise approximation of a straight-line path.

Excited by the favorable prospects of this new approach, I abandoned my frustrated attempts to precisely servo the robot's heading using two different proximity sensors, and reconnected their outputs in a logical-OR fashion as before. The desired steering bias was easy enough to implement, just by shifting the commanded steering angle over one position from centerline. Since there was an even number (i.e., 16) of discrete possible positions, there already was a slight bias towards one side anyway, since no commanded

angle corresponded directly to dead center. Accordingly, I used position 07 for a slight drift to the right and 08 for a drift to the left (i.e., full right was 00, and full left was 15), depending on which side the triggered sensor had been, and it worked like a charm! The new wall-hugging behavior was now stabilized with the unpredictable zig-zagging a thing of the past, and yet the reactive avoidance of discrete obstacles (i.e., versus extended wall surfaces) continued to function pretty much as before.

After this restructuring of the collision-avoidance strategy, the nature of the *Wander* behavior in general was much less random. It now tended to hug walls, resulting in a significant improvement in patrol efficiency. The robot spent much less time criss-crossing back and forth over the same area, opting instead for more natural routes of extended coverage, since walls tend to define perimeters. However, this efficiency increase was very application-specific; such an exploratory search pattern lends itself well to security, but would be much less desirable in the case of a cleaning robot. On the other hand, it would be very well suited to the task of mapping an unknown structure, once the localization issue had been suitably addressed, as would come to pass later with *ROBART III* (Pacis & Everett, 2004).

In retrospect, the development of this wall-hugging behavior was a significant milestone in the evolution of *ROBART I*, largely responsible for the continued growth of even more sophisticated navigational capabilities discussed later. The sensor hardware had not really changed all that much (other than viewing angles and gains); the real modification had been in how the software reacted to the sensor data. The key lesson here was that intelligent mobility required coordinated navigation and collision avoidance, and overly aggressive avoidance could easily impede successful navigation. Given my initial inspiration for all this had come from our rambunctious little housecat, I now feel rather badly in recalling how we banished him to "the farm" when it came time to depart Monterey for my next duty station.

5.1.3 Room Entry/Exit

I soon discovered that in solving the zig-zag problem, I had inadvertently created another, as is so often the case in robotics. The new wall-hugging behavior had an inherent tendency to bypass doorways, unless the latter were directly ahead in the path of travel. As a consequence, the robot was inclined to not enter rooms, preferring instead to traverse up and down the hallway, and on those rare occasions where it did venture into a room, it generally stayed there. Close examination of this emergent idiosyncrasy offered a fairly simple explanation, while simultaneously suggesting an equally simplistic mechanism for altering it.

The problem was the wall-hugging behavior combined with the *if-then rules* of *reactive collision-avoidance* (i.e., if blocked forward and to one side, turn in opposite direction) to keep the robot moving in a never-ending circuitous perimeter (Figure 5-3). Whichever way the robot turned when first encountering a wall would set the direction of loop flow (i.e., clockwise or counterclockwise), after which it had little reason or stimulus to change. The heuristic for escaping this trapped condition (in other words,

finding the door in order to exit the room) was simply to look for an opportunity to turn the other way.

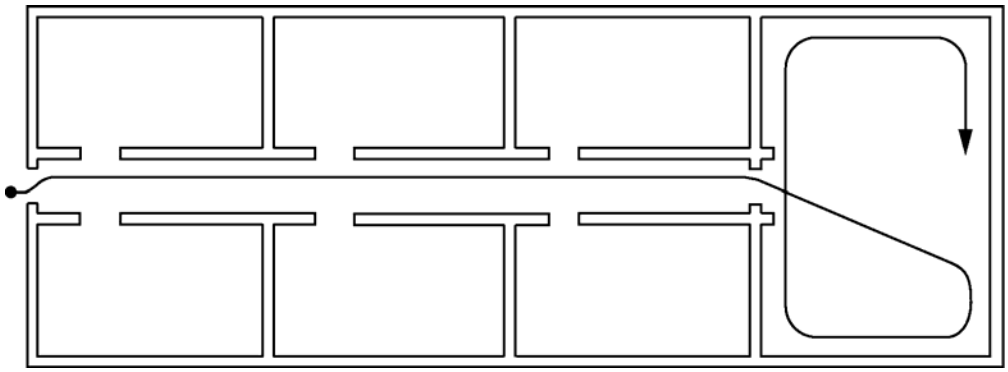


Figure 5-3. The new *wall-hugging behavior* combined with the existing *reactive-avoidance behaviors* to produce a circuitous perimeter traversal that generally precluded room exit.

Implementing this room-exit strategy involved three simple steps: (1) determining that such a trapped condition existed, (2) figuring out the turn direction associated with the circuitous loop, and (3) searching in the opposite direction for the doorway opening. The first two of these needs were addressed through the addition of two more count registers in the collision-avoidance interrupt routines. These registers (*left_turn_count* and *right_turn_count*) kept track of how many times the robot turned in the same direction in response to a blocked condition, and were both cleared every time the turn direction changed.

As a consequence, one register would always be zero, while the other was some value greater than or equal to one. The bigger the non-zero register value became, the more likely the robot was trapped in a circuitous path. The direction of turn was simply a function of which register had begun to increment. (I had intended to put another step in the program that cleared both registers if more than 30 seconds had elapsed between consecutive turns, but I'm not sure now almost 25 years later if I ever got to that stage.) The third step involved panning the head 45 degrees to whichever side the robot had been avoiding, and looking for a door opening with the head-mounted proximity sensor. An early demonstration of this hallway navigation scheme was performed for Katryn Pratt and Steve Rosen of television station KMST in Monterey, California, portions of which were aired on the station's "Sunday Edition" program on 30 May 1982.

It is somewhat interesting to note that today the iRobot® *Roomba* vacuum (Figure 5-4a) effectively achieves similar navigational results supported by the same simplistic-type sensors: tactile and short-range proximity. The designers at iRobot® have done a masterful job in optimizing performance, and were especially clever in selecting a circular low-profile design that can turn in place, which greatly facilitates getting unstuck from trap scenarios. This geometric configuration also allows the robot to have a preferred direction of motion along a wall surface, resulting in a counterclockwise perimeter traversal that favors the edge sweeper on the robot's right side. (If the *Roomba's* angle of approach to a wall geometrically suggests a subsequent clockwise traversal pattern, the robot simply pivots in place until a counterclockwise strategy is possible, as shown in Figure 5-4b.)

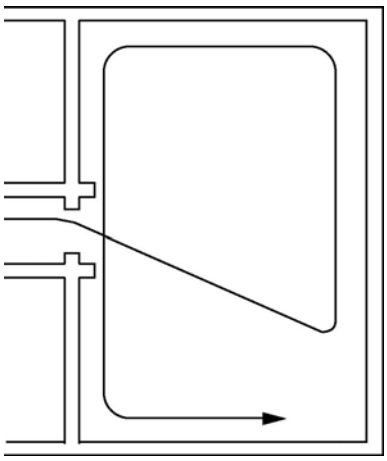
a)  b) 

Figure 5-4. a) Photo of iRobot's *Roomba* vacuum; **b)** diagram of wall intersect, showing the *Roomba's* pivot-left behavior to achieve a preferred counterclockwise direction of travel around the room perimeter.

Using convergent versus diffuse proximity sensors eliminates ambiguities associated with variations in surface reflectivity, allowing the wall-hugging servo loop to be tuned for almost straight-line motion. As in the case of *ROBART I*, the software knows when to terminate wall hugging and switch to an alternative behavior, such as an expanding spiral pattern for cleaning the room center. For these and other noteworthy innovations, the *Roomba* made history in 2002 as the first successful mobile robot to be mass marketed to the world. Over one million units had been sold less than 2 years later.

5.2 Model-Based Navigation

Obviously any robot intended to serve in a security role must be provided with a more intelligent means of navigating, as it would be no great challenge to outsmart a patrolling sentry that relied upon semi-random motion to carry it from place to place.

5.2.1 Absolute Model

One approach to a more effective navigational strategy involves building an absolute memory map in which the robot could encode relevant information about its environment as it moved about. I felt a simple way to do this would be to assign a single byte in memory to each square foot of floor space, as presented by Weinstein (1981) in the book *Android Design*. An average room 10 feet by 12 feet would require only 120 bytes, and thus an array representing a household floorplan could reside in less than 2 kilobytes of memory space. Over time, the robot could fill in the originally blank map with probability codes reflecting the chances of finding an object in any particular square. For example, a code of 0 could indicate that there had never been an object detected in that location. Code 1 might mean there was once but not always, code 2 indicating there probably is, and code 3 meaning there will always be an object in that square.

With a memory map of this type, I reasoned, algorithms similar to those used in computer games (such as *Othello*®, which had recently been released for the *SYM*) could be invoked to locate clear pathways through a room full of obstructions. The geographical position of the recharging station could be marked, as well as door openings, hallways, or even areas the robot should never traverse. For the security

application, optimal routes for patrolling could be preprogrammed or generated as needed by the software. As it turns out, all of these speculations would eventually come to pass in one form or another. On *ROBART II*, for example, a modified version of the A* planning algorithm (originally developed for use as a trace router in laying out printed circuit boards) was used to generate absolute path trajectories (Gilbreath & Everett, 1988).

The main problem with a modeling scheme of this sort is that the robot has to know its absolute location at all times, as well as its orientation in that spot, and in 1981 there was no inexpensive way to provide such a capability. Various means had been suggested, such as using sonar to establish the minimum range (and hence a perpendicular line) to each wall, and through geometry calculate position in terms of memory map coordinates. While this approach sounded plausible in theory, one glance around the average room reveals a significant number of drawbacks in the form of doorways, windows, and upright furniture, which would invalidate the sonar data and greatly complicate the needed software. The technique might work in an empty building with unobstructed wall surfaces, but it was clearly not practical under more realistic conditions. As already mentioned, the *LM1812* sonar performance in air left a lot to be desired.

5.2.2 Relative Model

Accordingly, I soon gave up on the *absolute-world-model* approach and began searching for a suitable alternative, something that could support a more-encompassing ability to anticipate and avoid obstructions. I wanted a bigger picture scheme than the current "tunnel-vision" perspective of *if-blocked-left-turn-right*, but I was somewhat at a loss as to exactly what this should be. Progressively adding more and more *if-then rules* to try and control such complex behavior can quickly get rather complicated, ultimately leading to hopeless confusion on the part of both the programmer and hence the robot.

Finite-state analysis is sometimes used to maintain more structure in circumstances where there are a fixed set of interdependent possibilities (i.e., states), but in early 1981, the concept of *finite-state machines* had not yet come to my attention. Rod Brooks and Jon Connell describe the use of distributed networks of augmented finite state machines in controlling mobile robots (Brooks & Connell, 1986), the most famous example of which is probably the six-legged robot *Ghengis* (Brooks, 1989). Joe Jones (2004) provides an excellent tutorial example of the finite-state-machine concept in his latest book, *Robot Programming: A Practical Guide to Behavior-Based Robotics*.

The field of mobile robotics is generally not that structured, however, due to a plethora of uncertainties introduced by the mobility aspect itself, so the number of possible states is not all that finite, except in the case of subsystems that look at some specific (hence bounded) subset of the overall picture. As a consequence, such problems today are often addressed through application of *fuzzy logic*, developed in the mid-sixties by Professor Lofti Zadeh of the University of California at Berkley (Miller, 1993), but not widely used until the late-eighties. John Holland (2003) would successfully apply the concept in his sonar-based navigation scheme for the Cybermotion® *Navmaster*, which we chose in

1990 as the optimal mobility platform for the MDARS-Interior robot (Everett et al., 1990).

Sadly lacking in such formal approaches, yet blissfully ignorant of my shortcomings in this regard, I once again fell back on careful introspection of how Mother Nature approaches the problem of situational awareness. We humans, I began to realize, somehow create a mental representation of where things are relative to ourselves, and this representation stays rather accurately registered even as we change position and orientation. Our peripheral vision subconsciously provides input on where surrounding objects are situated in the forward hemisphere of our travel. As these stationary objects pass behind us to either side, their visual representations in our brain are somehow replaced by virtual stand-ins that continue to flow in accordance with our relative motion. Our mind thus retains a reasonably accurate subconscious "feel" for where these no-longer-visible objects are located, in polar coordinates.

I found the nature of this "feel" rather intriguing, to say the least. It was as if some *potential-field* arrangement of neurons was firing in the brain, simulating discreet repelling forces that were spatially registered with the surrounding objects. We thus experience some nagging sensation in the back of our minds that something is in a certain position relative to our own, hence be careful not to collide. In discussing how the "brain keeps mental maps of nearby objects," John Ratey (2001) reports how Michael Graziano and his team at Princeton University performed experiments with monkeys that showed precisely this effect:

> "Graziano found that when the monkey's eyes locate an object, groups of neurons start to fire, and a subset of them continue to fire even after the object is out of sight."

Graziano's research in sensory-motor integration involved neurons in the ventral premotor cortex of monkeys (Graziano et al., 1997), which were shown to respond to the presence of ocular stimuli near the arms and face. An object placed within the visual receptive fields of such neurons caused them to fire, but more importantly, when the object was silently removed after first turning off the lights, a portion of these neurons continued to fire. This neurological phenomenon effectively allows the primate to "remember" the spatial locations of objects that can no longer be seen. Allan Hobson (2002) describes similar findings with regard to neurons in the hippocampus of rats in a maze:

> "Rats who are learning maze navigation will often show correlated firing of neuron pairs in the area of the brain known as the hippocampus, and this firing increases as they learn, as if neurons were making a physical link to represent the animal's orientation maps."

Returning now to my own experience, in addition to the angular cue, there also appeared to be an inverse correlation between the intensity of this warning sensation and the perceived distance to the object: the closer we think we are getting, the stronger the embedded warning becomes. I speculated at the time that our *conscious mind* collects the initial perceptual data from our vision to build this polar model, and then our

subconscious mind performs continuous updates to account for any subsequent movement on our part. (If we are already familiar with the surrounding layout from prior experience, this initial need for conscious perception is substantially reduced.) The *subconscious mind* later communicates the necessary spatial-awareness information to our *conscious mind* via this potential-field sensation just described.

You may have noticed this phenomenon yourself when playing sports. Take baseball, for example. The batter hits a high fly ball out towards your direction in left field, right down the foul line. You immediately give chase to your right, but the ball continues climbing, forcing you to take several steps backwards at the last minute. While you *consciously* squint against the sun and calculate the proper course of intercept, your *subconscious mind* reminds you of the outfield fence line to your rear, and the gopher hole you saw earlier off to the right. As you continue backing up, you actually feel a growing distraction that slows your advance, warning you of an impending collision. The closer you get to where your mind has envisioned the hazard, the more pronounced the effect, until it competes almost one-on-one with your conscious actions. Being somewhat of a wimp, you stop just in time, but unfortunately miss the ball as it sails a few inches over the top of the fence.

In deconstructing this scenario, it was *conscious perception* that created the initial mental image in your head of where things were around you, and your *subconscious perception* that maintained it, integrating over a period of time as you waited in left field for some action. Each time your team took the field you assumed the proper defensive position for the batter at hand, then made a mental note of your surroundings, including the locations of your nearby teammates. (In racquetball, it is called "court sense," and those players who do it well have far fewer injuries from slamming into surrounding walls.) Once the batter makes contact and the ball is headed your way, the *conscious mind* becomes focused on but one thing: getting into the optimal position in time to make the catch. Your *conscious mind* relies upon your *subconscious mind's* "situation map" to ensure you do not collide with an obstruction in the process. If it had to consciously multiplex between both tasks, you would forever play like an inexperienced beginner.

I was very intrigued by this observation, and sought to duplicate the concept as an interim solution, giving *ROBART I* the ability to generate a *relative* rather than *absolute* model of surrounding objects. By recording the angular position of obstructions seen by sensors mounted on the head, the robot could then examine this simplistic *memory map* for a conglomerate of state information previously unavailable. Instead of reacting only to the most immediate threat, the model-based avoidance strategy could now formulate more intelligent evasive solutions based on extended situational awareness of the entire forward hemisphere. Such a predictive approach would allow the robot to navigate in a far more effective manner until such time as better sensor and processing hardware for absolute localization became available.

Despite the perceived advantages, however, there is an inherent danger in building up state representations based upon questionably accurate sensor data, as pointed out by Jon Connell (1990):

> "Yet, to confidently add state to the system requires placing a lot of trust in the robot's sensory capabilities. Since later actions and perceptions are influenced by state, we want to verify that the remembered data is based on solid readings and is not just a sensory glitch."

I minimized this "erroneous-memory" problem by constantly flushing the state information saved in a very simplistic relative model, and replacing it each time with fresh data.

This *relative model* (Figure 5-5) required only 16 bytes of memory space, essentially one *state variable* for each of the points of resolution of the head-position A/D converter. Since a byte consists of eight bits, eight pieces of binary information could thus be stored for each pie-shaped sector of the robot's perceived world. From a navigational perspective, the most obvious "bit" of information needed would be a near-infrared return detected by the parabolic-dish proximity sensor, which had a range of 5 feet. The presence of any object within that distance would be recorded by setting the designated bit, say bit 0, in the memory location assigned to that specific head position. Bit 1 could represent the presence of a bright light source, bit 2 could be set if the passive infrared sensor output went high, bit 3 might correspond to a close sonar reflection from a transducer mounted on the head (strictly hypothetical, in that *ROBART I* did not have a head-mounted sonar), and so on.

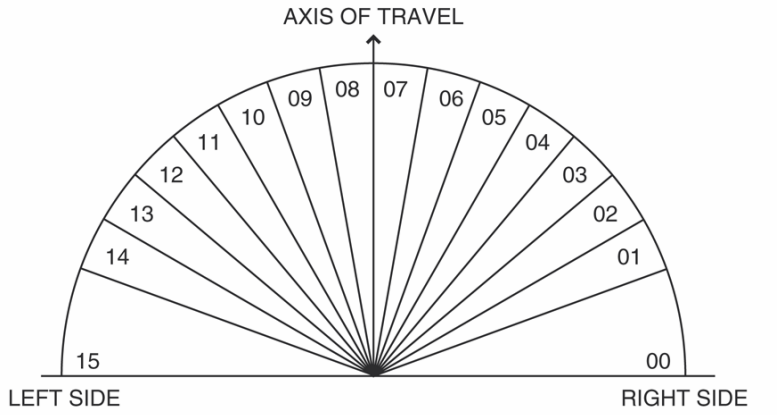


Figure 5-5. The relative world-modeling scheme on *ROBART I* assigned one byte of memory to each of 16 sectors representing the field of view of head-mounted sensors (adapted from Everett, 1982).

To keep the focus on navigation and avoid confusing the issue, only two such pieces of information were considered in my thesis discussion: (1) a near-infrared return from the parabolic proximity sensor (Figure 5-6), represented by setting the lower four bits; and (2) the presence of a bright light source, represented by setting the upper four bits of the appropriate memory location. The first step in the implementation involved writing code (i.e., subroutine *Survey*) to turn the head to position 00 (full right) or position 15 (full left), whichever was closer. The head was then swept through the full range of positions while the software polled the sensor inputs. If any were found to be active, the associated head position was read and appropriate bits set in the corresponding memory locations. Upon completion of the sweep, the state-variable data recorded in the model could be used in decision-making algorithms that dictated the robot's evasive actions.

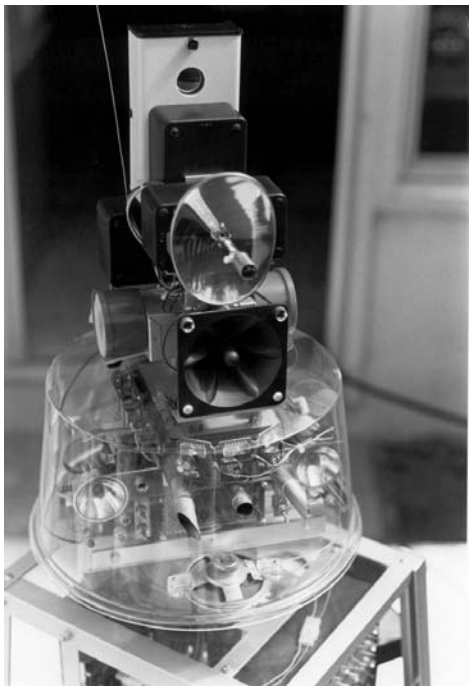


Figure 5-6. The parabolic reflector for the head-mounted proximity sensor is seen at front center, just below the passive-infrared motion detector (white enclosure) at the top of the photo.

One such algorithm was subroutine *Sort*, which examined the model to determine the starting and ending boundaries of any obstacle-free zones, expressed in terms of head position. This was followed by subroutine *Choose*, which selected the largest free zone, and subroutine *Center*, which calculated its midpoint. Since the model was implemented as a polar representation, this midpoint value could be directly used as a steering command for maneuvering the vehicle into uncluttered space, away from obstructions. It was also very useful in locating open doorways. As a result, the robot did not have to "stop and think" (i.e., calculate an evasive path from a Cartesian representation of its surroundings), but instead responded immediately in reactive fashion. This real-time response capability turned out to be an important key to success, given the rather stark limitations of onboard computational resources.

Figure 5-7 shows a sample printout of a test harness written during the development of these subroutines in July 1982. The memory contents following a sweep of the head *(Survey)* are listed, immediately below which are shown the starting and ending boundaries of the two biggest obstacle-free sectors. The larger of these sectors is then chosen *(Choose)* and its midpoint subsequently calculated *(Center),* as marked on the printout. Unfortunately, the time required to pan the head and update the contents of this relative model was 3.5 seconds, long enough for the robot's position and orientation to change considerably. Changes in position had minimal effect on the validity of data, due to the very low speed of advance (approximately 4 inches per second). Changes in orientation could be significant, however, given the possibility of rather sharp steering angles (up to 80 degrees). The prob1em was further complicated as the sweep could be

in the direction of the turn, or opposite to the turning direction, resulting in invalid data in either case.

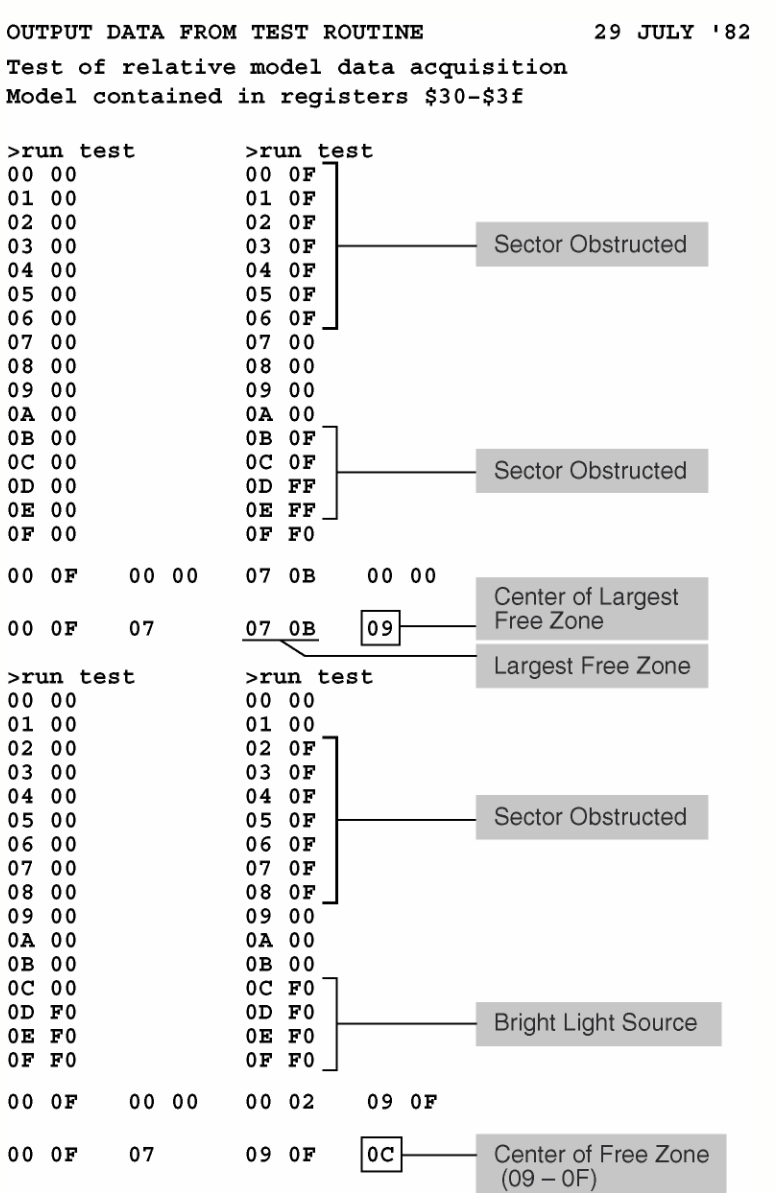


Figure 5-7. The results of four actual test runs conducted during development, showing the sequential results of subroutines *Survey*, *Choose*, and *Center* processing real sensor data during a sweep.

As previously discussed, this high probability of invalid data was a significant concern since the *relative-model state* would directly influence any subsequent navigational planning behavior. The primary source of error, however, was not so much a function of inaccurate sensor data, but more to do with acquisition lag associated with pan-motor velocity, which was limited by the chosen hardware. So, to reduce the data acquisition time, and thus minimize the error, a more practical scanning approach was to limit the sweep to those sectors directly in the robot's path, as opposed to the entire domain of 16 sectors.

Since the sweep limits were determined by software, this adjustment was easily implemented. While the robot moved forward, subroutine *Radar* would pan the head back and forth between sectors 06 and 09. If an object came into view on the left, the robot's heading was adjusted to the right, and vice versa. Should the entire four-sector region become blocked, the platform came to a halt and performed a complete sweep in an attempt to locate a clear path. Since the time required to update this abbreviated model was significantly less than 3.5 seconds, and turn angles were intentionally kept within 40 degrees as opposed to 80, the chances of loading invalid data were greatly reduced. While this compromise increased the data update rate to a minimally useful level, I was convinced that future designs needed a staring array of collision-avoidance sensors for optimal reactive response, or orders-of-magnitude faster scan rates.

In the meantime, however, what was needed was a means for the navigational loop to know when an interrupt had occurred, so that the old data in the model could be flushed and updated. Since this requirement had been anticipated at the time the interrupt routines were written, the solution was relatively simple. Register *IRQ.num* kept track of the number of collision-avoidance-related interrupts, incremented each time the IRQ routine was executed. If this register was checked at the beginning of an abbreviated sweep, and its value was the same upon completion of the sweep, then the data recorded during the sweep were assumed valid, as no interrupt occurred to distract the CPU.

If the value was not the same, then the model was cleared, indicating no obstructions were present. This may not have been the case, but the situation was soon remedied when the next sweep was performed, provided another interrupt did not take place to invalidate it also. The robot moved straight ahead during the interim. If another interrupt did occur, the collision-avoidance routine would take over anyway to skirt the obstruction. Once clear, the software returned to the navigational loop and the sweep resumed.

This approach allowed the collision-avoidance "layers" to work together without conflict, with the longer-range infrared sensor planning ahead, yet yielding to the close-range proximity detectors when a collision threatened, and ultimately to the tactile sensors, should an impact actually occur (Table 1). With this new ability to look out and see obstacles 4 to 5 feet ahead, the robot could usually alter course in time to pass to one side. If it became boxed in, it could stop and scan the entire range of head positions for a clear zone. This upgrade offered significant improvement over the purely random motion of *Wander*, but still left a lot to be desired for patrols to be made in an orderly fashion, in that it basically represented a collision-avoidance behavior only. For more effective autonomous motion, some basic localization (i.e., referencing) capability was required.

In reality, the height of the head-mounted long-range proximity sensor caused it to miss many smaller obstacles below the beam axis, which further strengthened my bias towards a staring array of distributed sensors providing full volumetric coverage in the direction of intended motion. It also made me realize that obstruction clutter was most pronounced down low and grew progressively less severe with increasing height. In other words, our dependence upon gravity to keep things in place means there typically

are a lot more miscellaneous objects near the floor than up towards the ceiling, and so the robot's sensors should be arranged accordingly. Scanning ahead for obstacles only at the 4-foot level was clearly problematic.

LEVEL	BEHAVIOR	RESULTING ACTION
Higher	*Radar*	Look ahead for encroaching obstacles
	Survey	Look for opening in forward hemisphere
	Dock	Home in on recharging station
Intermediate	*Wander*	Seek clear path along new heading
	Wall Hugging	Follow adjacent wall in close proximity
Lower	*Get Unstuck*	Escape a trapped condition
	Proximity Reaction	Veer away from close proximity
	Impact Reaction	Veer away from physical contact

Table 5-1. The navigation scheme on *ROBART I* provided a layered hierarchy of behaviors that looked ahead for a clear path, reactively avoided nearby obstacles, hugged extended planar surfaces, and responded to actual impacts. A basic tenet of this approach was the ability of certain high-level *deliberative behaviors* to influence or even disable the intermediate and lower level *reactive behaviors*.

5.2.3 Basic Localization

Augmenting *Wander* with *wall-hugging* and *scan-ahead* behaviors had been a major step in the right direction, but it still fell short of addressing the "from point A to point B" requirement. Again, what was ultimately needed was a means of accurately determining absolute position and orientation. Until such time as this could be practically implemented, however, *ROBART I* had to learn to navigate with the information it had available, which meant finding some simplistic means of "grounding" the relative model to the household floorplan.

In the beginning, the robot simply referenced off the position of its recharging station, which it could locate and positively identify out to some finite distance. As a consequence, it tended to stay close to the charger (very much like a young bird lingering near the nest as it is learning to fly), so as not to go beyond effective beacon range (roughly 12 feet or so). Such a constraint was way too limiting for my envisioned purposes, however. I needed a simple yet reliable scheme that facilitated further exploration, while ultimately ensuring a safe and timely return to the energy supply.

Hallways, I reasoned, should be relatively easy to recognize, being long and narrow. Once in the hallway, the *wall-hugging* behavior would ensure a certain repeatable structure to the robot's motion. If the household floor plan allowed for positioning the recharging station near the hallway entrance, then the robot could find the hallway and proceed down it, stopping at each doorway to check adjoining rooms while on patrol. This arrangement allowed for synergistic combination of two primitive behaviors, *wall-hugging* and *beacon-tracking*, resulting in a major increase in navigational capability. When it was time to recharge, the robot simply turned around and followed the walls back down the hall until it was within range to reacquire the recharging beacon. Preliminary experiments showed great promise, and so this technique became the foundation for the *hallway navigation scheme* to be discussed in the next section.

5.3 References

Brooks, R.A., and Connell, J., "Asynchronous Distributed Control System for a Mobile Robot," Proceedings, SPIE Symposium on Optical and Optoelectronic Engineering, Vol. 727, Cambridge, MA, pp. 77-84, 1986.

Brooks, R.A., "A Robot That Walks: Emergent Behavior from a Carefully Evolved Network," *Neural Computation*, Vol. 1, No. 2, pp. 253-262, 1989.

Connell, Jonathan H., *Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature*, ISBN 0-12-185230-X, Academic Press, San Diego, CA, 1990.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot*, Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982.

Everett, H.R., "A Second-Generation Autonomous Sentry Robot," *Robotics Age*, pp. 29-32, April, 1985.

Everett, H.R., Gilbreath, G.A., and Holland, J.M., "Hybrid Navigational Control Scheme," SPIE Mobile Robots V, pp. 291-298, November, 1990.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, A.K. Peters, Ltd., Wellesley, MA, June, 1995.

Ferrel, C.L., "An Autonomous Mobile Robot, a Planetary Microrover," *Sensors*, pp. 37-47, February, 1994.

Gilbreath, G.A., and Everett, H.R., "Path Planning and Collision Avoidance for an Indoor Security Robot," Proceedings, SPIE Mobile Robots III, Cambridge, MA, pp. 19-27, November, 1988.

Graziano, Michael S.A., Hu, Xin T., and Gross, Charles G., "Coding the Location of Objects in the Dark," *Science*, Vol. 277, Issue 5323, pp. 239-241, 11 July, 1997.

Hobson, J. Allan, *Dreaming: An Introduction to the Science of Sleep*, ISBN 0-19-280304-2, Oxford University Press, Oxford, UK, 2002.

Holland, John, *Designing Autonomous Mobile Robots: Inside the Mind of an Intelligent Machine*, ISBN 0750676833, Newnes, 2003.

Jones, Joseph L., *Robot Programming: A Practical Guide to Behavior-Based Robotics*, ISBN 0-07-142778-3, McGraw-Hill, New York, NY, 2004.

Kaneko, M., "Active Antenna," IEEE International Conference on Robotics and Automation, San Diego, CA, pp. 2665-2671, May, 1994.

Miller, Daniel P., "Putting Fuzzy Logic in Focus," *Motion Control*, pp. 42-44, April, 1993.

Pacis, Estrellina, and Everett, H.R., "Enhancing Functionality and Autonomy in Man-Portable Robots," Proceedings, SPIE Unmanned Ground Vehicle Technology VI, Defense and Security, Orlando, FL, 12-16 April, 2004.

Ratey, John J., *A User's Guide to the Brain: Perception, Attention, and the Four Theaters of the Brain*, ISBN 0-679-45309-1, Pantheon Books, Random House, New York, NY, 2001.

Rosenblum, L.D., Gordon, M., and L. Jarquin, "Echolocation Distance by Moving and Stationary Listeners," *Ecological Psychology*, Vol. 12, No. 3, 2000.

Update, "Robotics on the Battlefield," *Robotics Update*, Volume 4, Number 2, Space and Naval Warfare Systems Center San Diego, CA, Summer, 2004.

6
Hallway Navigation

"If an idea's worth having once, it's worth having twice."

Tom Stoppard

I always felt like the hallway navigation scheme I had proposed (but never fully described) in my thesis (Everett, 1982) had considerable potential as a stop-gap improvement, given my less-than-optimistic assessment of any near-term breakthroughs in absolute localization. I had very little to do my last quarter in graduate school, having finished my thesis early, which was mostly because I started working on the project before the first quarter even began, 2 years previous. Once the manuscript was turned in, I played around quite a bit with the implementation of such a concept, but never really documented it. Part of the problem was I had already begun the physical construction of *ROBART II* (Figure 6-1), which had by this point taken over my real focus.

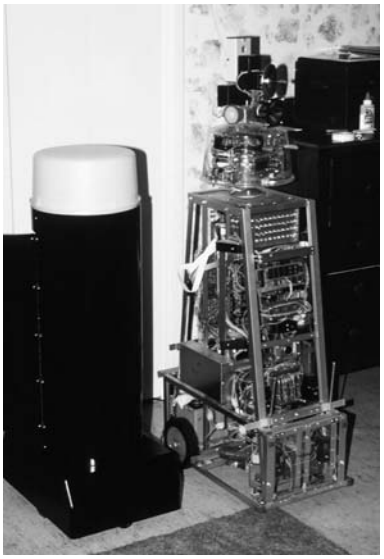
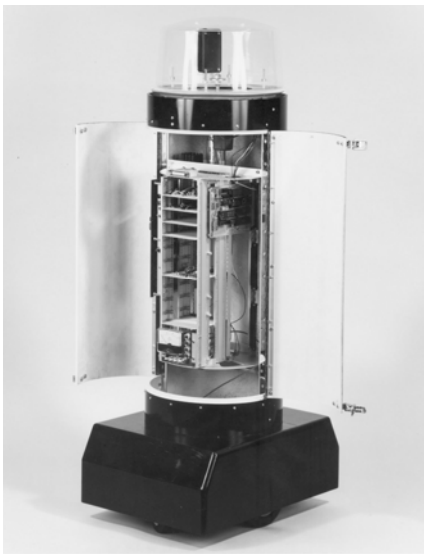
a)  **b)** 

Figure 6-1. a) A 1982 photo of *ROBART II* (left) in Monterey, CA, with a *Tupperware*® bowl serving as a temporary head; **b)** rear view showing the computer card cage in the very early stages of development. The mobility base in both photos is a wooden mock-up, later used as a form for the fiberglass housing.

But as anyone who has tried it will readily attest, there is a distinct time lag in robotics between the start of new construction and that ellusive point later on where it becomes possible to actually write some software. Meanwhile, I continued to develop code on *ROBART I*, although I had already dismissed it in my mind as a dead-end antiquity. I envisioned three navigational scenarios that needed to be addressed in order of increasing complexity:

***Route Retrace* –** A "breadcrumb" ability that allowed an exploring robot to find its way back to the recharging station.
***Structured Patrols* –** A collection of learned paths (i.e., from above exploring) that could be assimilated into more effective patrol routes.
***Operator Dispatch* –** A means for human-directed transit (i.e., go to a designated location).

I actually got quite a bit working in the first category (*Route Retrace*), albeit somewhat site-specific, but there were still a few bugs to be ironed out when it came time to pack it all up and move to my next duty assignment on the east coast. The floor plan of my new two-story house in northern Virginia turned out to be decidedly lacking in terms of hallways, which put the final nail in the coffin. Accordingly, *ROBART I* was loaned to the fledgling robotics group at the Naval Surface Weapons Center in White Oak, MD, entrusted to the watchful care of an MIT co-op student by the name of Anita Flynn, now a famous pioneer in the field of micro-robotics (Flynn et al., 1989; Hollar et al., 2003).

Figure 6-2. Anita Flynn of the MIT AI Lab, as a co-op student with *ROBART I* at the Naval Surface Weapons Center, White Oak, MD, 1984.

Years later, I would partially assuage that lingering nag of unfinished tasks by writing an article on the subject for a hobbyist-oriented magazine called *The Robotics Practitioner*, which sadly is no longer in circulation. Entitled "Autonomous Navigation on a Shoestring Budget" (Everett, 1996), it described a practical navigation scheme that might appeal to up and coming robotics enthusiasts with limited experience and financing. I have reproduced (and in some cases expanded) the relevant sections of that document in this section for any interested reader who may want to recreate such a strategy on an experimental platform of his or her own design.

6.1 Simplistic Global Navigation

This section discusses a primitive global-navigation solution that can be easily implemented on an 8-bit microprocessor at minimal cost. The approach is insensitive to the type of drive and steering configuration, and does not even require a dead-reckoning capability of any sort. The more complex general case is effectively bounded by limiting operation to indoor environments with easily traversable floor surfaces, and further constrained to a series of rooms arranged in ordered fashion along either side of a hallway.

6.1.1 Background

The hallway-navigation scheme on *ROBART I* tracked its general location along a linear path, knowing only that the robot was somewhere between predefined nodes in a topologically connected network, and moving in a known direction. This is the underlying concept behind the age-old *Automatic Block System* that electrically monitors the whereabouts of freight and passenger trains in the United States, essentially a level-triggered absolute encoding scheme with very low resolution. A line of railroad track is segmented into a series of electrically isolated segments, or *blocks*, each roughly safe-braking-distance in length. The left and right rails within a block are shunted together by the presence of a train, thus activating color-coded signaling equipment to alert other engineers that the track ahead is in use, and to automatically lower the crossing barriers at roadway intersections.

The *Automatic Block System* works with minimal sophistication simply because the train has to follow the tracks, and in doing so alters the electrical properties (i.e., continuity between rails) of the occupied block in an easily detected manner. Unfortunately, there is no comparable structure in hallways giving rise to an inherent sensing modality for absolute position, so how then can the robot know what "block" it occupies at any given time? The most obvious hallway feature that humans exploit in goal-driven navigation is the presence of doorways. We can instantly direct a first-time visitor to the restroom in our home, for example, with the following concise yet unambiguous phrase: "Down the hall, second door on the right." In larger structures (i.e., such as hospitals, hotels, and office buildings), we often are further aided by room identification signs associated with numbered doorways, perhaps even a central directory. *ROBART III* would later learn to make use of this supporting visual information (Everett et al., 2004), but I had to keep things as simple as possible in 1981.

Accordingly, *ROBART I* employed an edge-triggered *incremental* encoding scheme (versus absolute) to keep track of its relative position in the hall, based on the perceived transition from diffuse wall surfaces to open doorways. To assist in first finding the hallway, the recharging beacon was suitably positioned in a known location as shown in Figure 6-3. Once in the hall, the robot would move parallel to the walls in reflexive fashion, guided by the near-infrared proximity sensors. General orientation in the hallway could be determined by knowing which direction afforded a view of the beacon. With *a priori* awareness of where the individual rooms were situated with respect to each other along this hallway, *ROBART I* could relocate to any desired goal location by counting off the correct number of openings on the appropriate side of the hall.

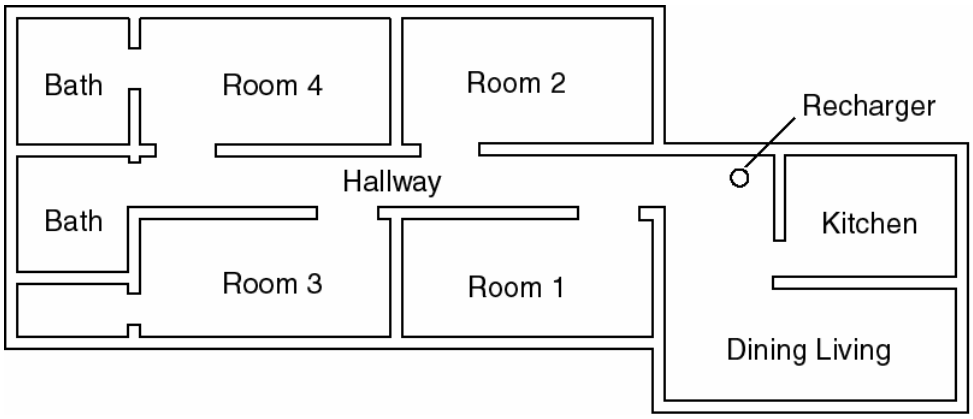


Figure 6-3. Floor plan of the operating environment for *ROBART I* in Monterey, CA, showing the location of the recharging station (small circle) at the right end of the hallway.

Each room was assigned a number, with odd-numbered rooms on the left in increasing order when moving away from the beacon (see again Figure 6-3). A dedicated behavior primitive took care of the specifics associated with guiding the robot through the detected door opening into the room of interest. Once inside a particular room, the robot's *Wander* routine would tend to hug the wall, traversing a rectangular path which eventually brought it back to the open doorway. By again invoking a doorway penetration behavior, the robot could re-enter the hallway and turn in the direction of the next room to be visited. The key assumption here was that each room had only one open door, which is fairly reasonable for a typical household environment, provided the closets are kept shut.

ROBART I thus knew in which room it was located at any given time, but not its absolute *X-Y* position or heading within that room. In a global sense, this could be regarded as operating from a *relative* navigational map as opposed to an *absolute* map depicting the actual floorplan. Interestingly enough, we humans tend to function in much the same fashion, in that we know where objects are with respect to one another, and have but a fuzzy feel for their absolute representation. When entering a darkened kitchen for a late-night snack, for example, we reach instinctively to a certain location for the light switch, *relative* to the doorway of entry. Such a *relative representation* makes it much easier to retain the necessary information in memory for a large number of locations. Imagine how cluttered our brains would get if we were forced to store an absolute model of everything we observed. The same limitation applies to an 8-bit microprocessor with but 64-kilobytes of address space, only more so.

6.1.2 Making It Happen

To replicate a similar navigational capability today on a mobile platform of your own choosing, only a few simple behavioral primitives must be addressed:

Wander – Causes the platform to move forward, turning away from obstructions.
Find Door – Locates an open doorway on the specified side of the robot.
Enter Door – Reflexively guides the robot safely through the detected doorway.
Verify Direction – Confirms the robot is traveling in the proper direction (optional).

In addition, some means of establishing the relative relationships of the rooms is required, such as a linked-list topological representation, or even more simply, a lookup-table data structure as was implemented on *ROBART I*. A couple of state variables are also needed to tell the robot which room it currently occupies, and in the case of the hallway, the direction of motion (i.e., in terms of increasing or decreasing room numbers). The following sections discuss the actual implementation of these basic behavior primitives, followed by one possible approach to recreating such a relative hallway-navigation scheme today.

Wander

The collision-avoidance sensing needs of a simple *Wander* routine can be met with three optical proximity sensors arranged in a fan-shaped pattern as depicted in Figure 6-4. The Banner Engineering *Valu-Beam® SM912D* diffuse proximity sensor (Banner, 1993a, 1993b) shown in Figure 6-5 performs well in this mode, draws only 40 milliamps at 12 volts DC, and provides a simple binary (i.e., target/no-target) output line for ease of interface. A gain-adjustment potentiometer is provided to set the maximum effective detection range for typical diffuse target surfaces (such as a wall) anywhere out to about 8 feet. Detection setpoints of 12 inches for the left and right sensors and 18 inches for the center sensors are good starting points, but you should experiment for best results. The cost of this unit is a bit high by most hobbyists' standards, about $80, but the payback is immediate availability, a simplistic binary interface, and reliable operation.

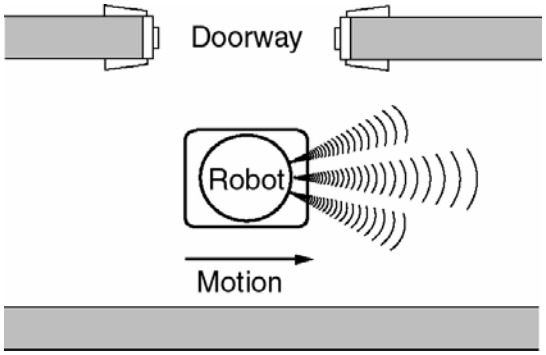


Figure 6-4. Three optical proximity sensors, three sonar sensors, or the superposition of both can be used to implement a simple *Wander* routine. Maximum effective detection range for the center unit should be just a bit further than for left and right.

Joe Jones and Anita Flynn (1993) describe a method for making your own low-cost proximity sensor with a somewhat shorter effective range, based on the Sharp® *GP1U52X* near-infrared detector module. Similarly, McManis (1995) presents details for building a multi-zone proximity sensor using the Sharp® *IS1U60* detector. More recently, Sharp® has introduced a new line of optical sensors that determine range from triangulation, and thus are insensitive to variations in surface reflectivity (Acroname, 2005). The *GP2D12* model (Figure 6-5 right) produces a (0- to 3-volt) analog output over an effective range of about 4 to 32 inches, with a fairly tight beam pattern for good angular resolution, and draws only 25 milliamps at 5 VDC. As with all triangulation

ranging sensors, the output is non-linear, with resolution degrading as a function of distance.

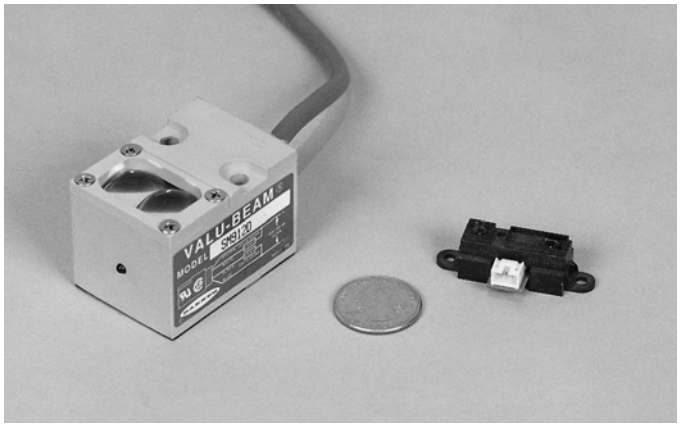


Figure 6-5. Banner Engineering's *Valu-Beam® SM912D* diffuse proximity sensor (left) has an adjustable gain control to set the maximum detection range to a diffuse surface, such as a wall; the Sharp® *GP2D12* rangefinder (right) has a maximum range of about 32 inches, and draws only 25 milliamps at 5 VDC.

Alternatively, three ultrasonic transducers can be multiplexed to a single Polaroid® ranging module (Everett, 1995a) to achieve similar effect, but the interface requirements are a little more complex. (As a further option, self-contained sonar sensors are available with analog outputs.) The highest probability of obstacle detection is of course achieved with redundant coverage from both optical and ultrasonic sensors. For such dual-mode implementations, the polling software can logically OR the outputs together for any sensors aligned along a common axis (i.e., looking in the same direction). In this fashion, if *either* a sonar *or* an optical sensor reports an obstruction in a particular direction, the robot will turn away. For sake of simplicity, however, the remainder of this discussion will be limited to a basic configuration involving three optical proximity sensors only.

The simplest implementation of the *Wander* algorithm is probably rule-based, taking the form of a series of conditionals that alter the robot's direction of travel in response to potential obstacles detected by the forward-looking sensors. Obviously, if the right-hand sensor sees a target, the robot turns left, whereas if the left-hand sensor sees something, the robot turns right. If the center sensor also sees the target, the rate of turn is increased. However, if the center sensor only sees a target, the robot should turn either left or right in accordance with a preset variable (*turn_preference*). Finally, when all three sensors are detecting (i.e., path completely blocked), the robot should pivot in place (for a differentially steered platform) or back up slightly before turning (in the case of tricycle or Ackerman steering).

These predefined response actions should continue for some finite time interval (typically 0.5 to 2 seconds) after the condition clears to ensure stability. When sensor gain, turning rate, and delay parameters are appropriately tuned through experimentation, this limited amount of hardware can be used to implement an effective wall-hugging motion that guides the robot safely down the hallway. Similarly, prespecifying a pivot direction for the blocked condition causes the platform to traverse the perimeter of a room in a rectangular pattern, either clockwise or counterclockwise, depending on the current value of *turn_preference.*

Find Door

The *Find Door* behavior requires some physical means for locating an opening on either side of the hallway. This task can be readily accomplished with a pair of side-looking optical proximity or range sensors set for a maximum detection distance of about 4 feet. Alternatively, two side-mounted sonar transducers can be used for this purpose. Optical sensors have a bit of an advantage in that they can be aimed forward just a bit to give advance warning in time to turn, whereas sonar sensors work best when the beam is kept nearly orthogonal to the target surface, due to problems associated with specular reflection (Everett, 1995a; 1995b). If your robot has a positionable head (i.e., pan axis), you can get by with a single sensor and mechanically point it in the desired direction. While this was the approach followed on *ROBART I*, it is much simpler and faster to electronically multiplex two (or more) sensors.

Enter Door

The *Enter Door* primitive requires no additional sensor hardware. Once the doorway is detected by *Find Door*, a turn is initiated in the proper direction. A slight delay might be required first to achieve the proper starting point, but this can easily be determined through trial and error. The actual rate of turn must also be optimized for the dynamics of your particular platform. The secret to success here lies in the fact that the *Wander* behavior is not inhibited during execution of the *Enter Door* routine. If the forward-looking sensors supporting *Wander* detect a target, the *Enter Door* routine terminates and *Wander* takes over. The reflexive-avoidance behavior of *Wander* adjusts the platform's direction of motion to keep it from impacting the sides of the doorway, the door itself, or an adjoining wall. The result is seamless and deceptively intelligent continuous motion.

The high-level software determines which way the robot should turn when re-entering the hallway by checking to see if the current room number is odd or even, comparing the next room number on the visit list to the current room number, and then setting the *turn_preference* register accordingly. (We will address this in more detail later.) The bigger challenge is deciding when to make the turn. If no doorways directly oppose each other in the hallway, the solution is rather trivial, as the opposite wall (Figure 6-6) will force *Wander* to turn at a distance of approximately 18 inches when the center proximity detector picks up a reflection. The pre-specified value stored in the *turn_preference* register ensures that the robot turns in the correct direction.

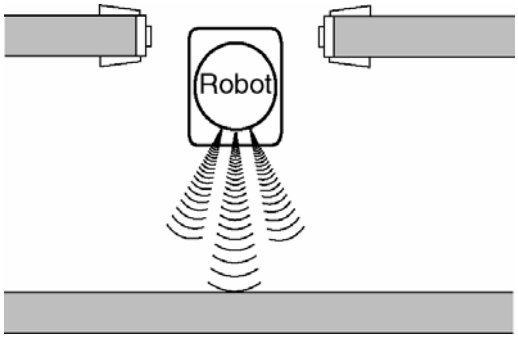


Figure 6-6. Upon re-entering the hallway, the robot turns in the pre-specified direction stored in *turn_preference* when the center proximity sensor detects the opposite wall as shown above.

If there is another open doorway directly across the hall, however, such a quick and dirty solution fails, for obvious reasons. In this more complicated scenario, you must use the two side-looking proximity detectors to determine when the robot has passed through the doorway and into the hall, and then force a turn. You can temporarily postpone solving this dilemma by simply closing off one of the two doors and eliminating that room number from the linked-list representation, to be discussed later. The idea here is to start simple and then add smarter routines as you become more experienced. You can learn a lot about sensor interaction with various target surfaces by observing your initial prototype trying to cope with the real world, and this valuable experience will provide significant insight into how to improve your design.

Verify Direction

The *verify direction* primitive (see Table 6-1) allows confirmation that the robot is indeed properly oriented in the hallway (i.e., moving in the direction of increasing or decreasing room numbers). While not absolutely necessary, this feature adds significantly to the robustness of the relative navigation scheme. On *ROBART I*, this task was accomplished by activating the optical beacon on the recharging station and checking to see if the robot could detect the results (i.e., was the platform facing the beacon?). This technique was available at no extra cost since the beacon tracking hardware was already in place to support automatic recharging, but this will vary from one system to another.

Value	Meaning
00	robot not in hallway
01	decreasing room numbers
02	increasing room numbers

Table 6-1. Range of possible values for the state variable representing the robot's direction of motion in the hallway.

A much simpler and more effective solution is now possible in the form of a low-cost electronic compass. While the more expensive models are accurate to within a couple of degrees, even the very cheapest versions (i.e., around $40) are more than sufficient to eliminate a 180-degree ambiguity in heading. Chapter 12 in *Sensors for Mobile Robots* (Everett, 1995b) describes the principle of operation and interface requirements for these devices in detail.

Topological Map

The most expeditious representation of the room inter-relationships is probably in the form of an array data structure as illustrated in Table 6-2, which can be easily implemented on an 8-bit microprocessor using the indexed addressing mode. The index serves as a pointer to the current room number, and by decrementing or incrementing the index accordingly, you can step through memory in the same order the robot would encounter the rooms as it traverses down the hall. Checking to see if the destination room number is odd or even determines on which side of the hall the doorway will be found.

Upon first glance it may seem like the array structure is not even necessary, as the data value merely echoes the index value, and a simple variable *current_room* would suffice. For an ideal scenario such as depicted in Figure 6-3, that is indeed correct. The use of an array, however, allows additional information about each room to be encoded in the upper nibble, while the lower nibble represents the room ID number. For example, the simplistic case illustrated above assumes there are an equal number of rooms on both sides of the hall, and repetitive increments of the index will step through memory in precise correlation to the robot's passage by each. In reality, such may not be the case, as rooms can vary in size.

Address	Value
XX00	00
XX01	01
XX02	02
XX03	03
XX04	04
XX05	05
XX06	06

Table 6-2. A one-dimensional-array data structure contains all the necessary information to describe the room inter-relationships in support of the relative navigation scheme. The upper nibble of the data value can be used to encode special information, while the lower nibble reflects the assigned room number.

An easy way to preserve the validity of the algorithm in this situation is to allow for the representation of *phantom rooms* that subdivide the space occupied by a single large room into the appropriate number *n* of smaller rooms. Since *n-1* of the *phantom rooms* will not have detectable doorways, they are specially marked by setting a designated bit in the upper nibble so the algorithm will increment the index without waiting for actual doorway detection. Other bits in the upper nibble can be used to mark rooms that have doorways facing another door directly across the hall, rooms containing battery recharging facilities, or perhaps even rooms that lead to other rooms.

The basic algorithm (i.e., not yet enhanced to accommodate *phantom rooms*) is presented in flowchart format in Figure 6-7. If the robot is exiting room 1 and wants to go next to room 4 (see again Figure 6-3), a left turn is in order upon entering the hallway (i.e., since the current room is odd and 4 > 1). The index is initially pointing to room 1. The algorithm first resets the *direction_of_motion* flag, then waits until the robot is traversing down the hallway with a wall detected on both sides (to avoid mistaking the doorway just exited for the anticipated detection of room 2).

After this hallway-verification condition is achieved, the right-hand doorway detector should next pick up the opening that marks the entrance to room 2. The room index is incremented, but does not yet match the destination room number, so the software loops back up and waits for doorway 2 to clear. After walls are again detected on both sides of the robot, the left-hand detector is monitored for the opening at room 3, and so forth. When the room index finally matches the destination room number, the algorithm calls the *enter_door* routine and turns the robot left or right, based on whether the room to be entered is odd or even, and the current direction of motion.

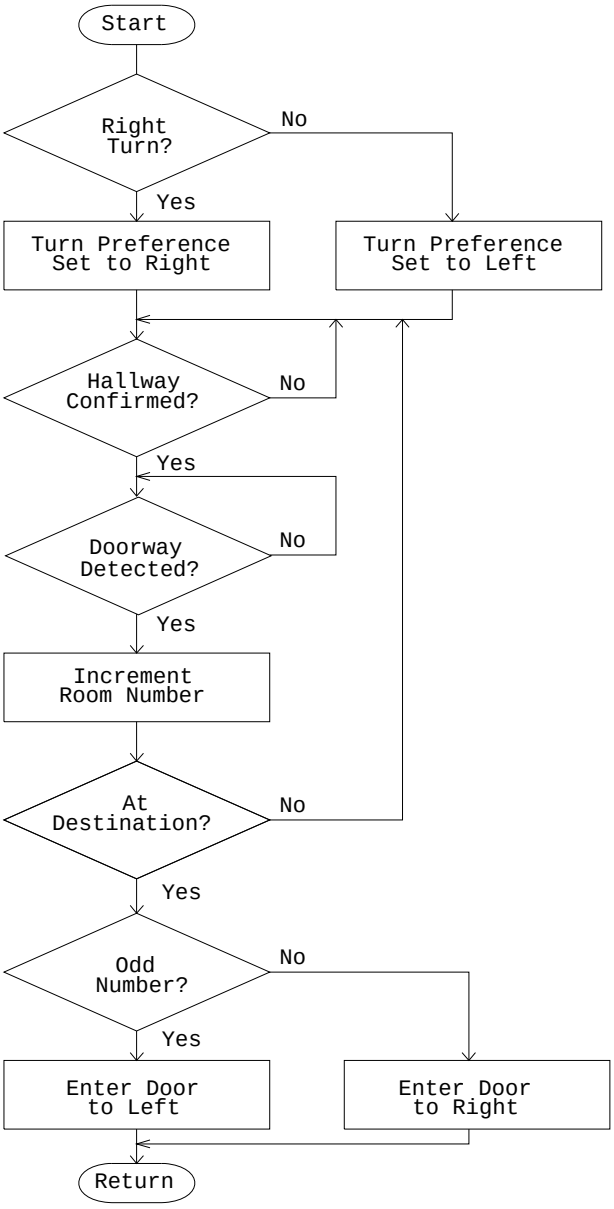


Figure 6-7. Flowchart of the basic hallway navigation algorithm partially implemented on *ROBART I.*

6.2 Helpful Hints

The single most important rule is to start simple and not try to solve all possible problems with your first design. You do not, for example, have to be able to navigate in every room in the house right off the drawing board. Select a couple of representative rooms along the hallway that lend themselves well to the concept, and get the system up and running. Watch what happens and go from there. Once you have your control algorithms stabilized through optimal turn rates, velocity profiles, and sensor gain settings, you can start adding to the robot's overall intelligence through incorporation of smarter algorithms. Observe the inevitable failure modes and determine what made the system do what it did, then adjust accordingly.

The left and right collision-avoidance sensors should be angled out just enough to pick up the wall surface when the robot gets to within about 4 to 6 inches, running parallel to the wall. If the fan-out angle is too large, the robot may zig-zag instead of hugging the wall, and there may also be some blind spots in the forward collision-avoidance coverage. An angle of 20 to 25 degrees from center should work fairly well. Zig-zag motion will also result if the avoidance reaction is too severe, due either to excessive turn rate or too much execution time. Experiment a bit and adjust as needed.

The maximum range of the doorway-detection sensor should be set at a greater distance than will be observed under worst case conditions in the hallway, yet not enough to pick up a distant target inside the room. Again, 4 feet is a recommended starting point for a 3-foot hallway, but you may want to increase it a bit if you plan to scan the sensor back and forth in search of a doorway from the inside of a room. It is best to "debounce" the sensor outputs through low-pass filtering to eliminate erroneous readings. In other words, require the sensor to show an opening for *X* number of consecutive reads, with a slight delay each pass through the loop, before reacting to the data. This way, if a momentary loss of target occurs due to specular reflection, the robot will not turn prematurely. When an actual doorway is present, the sensor output will toggle and remain in the new state for an appreciable length of time. If you delay too long before turning, however, you'll overshoot the door.

Note that room doors usually swing inward and almost always are situated in a corner so the door folds back 90 degrees against an adjoining wall (Figure 6-8). This arrangement allows for more efficient use of wallspace and makes the door easier for humans to shut, relative to a door that folds back 180 degrees against its own wall. (It also facilitates detecting closed doors along a hallway by using an appropriate ranging sensor, as will be discussed later.) Such a structured relationship can be exploited to a certain extent when attempting to re-enter the hallway, in that the door itself can be used to trigger a turn through the open doorway. For example, if the robot entered Room 1 in Figure 6-3 and made three right turns as it followed the wall perimeter, it would then be approaching the open doorway as shown in Figure 6-8. By resetting *turn_preference* at that point to *left*, the robot would exit through the open doorway back into the hall.

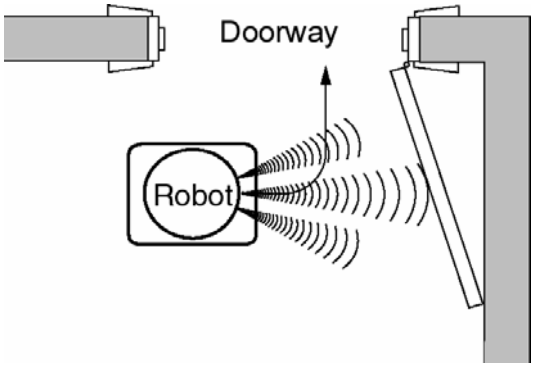


Figure 6-8. A reflexive avoidance maneuver (preset for a left turn) triggered by detection of the open door can be exploited to facilitate room exit through the adjacent doorway.

In the real world, however, most rooms are not clutter-free as implied above. If you have problems locating the doorway in order to exit a room, you may want to experiment with *polarized retro-reflective* optical sensors such as the Banner *Q85VR3LP*. The advantage of this type over the *diffuse* variety is unambiguous detection of a retro-reflective tape that can be attached to doorway edges for positive identification. This type of proximity sensor will not respond to diffuse target surfaces such as walls, or even direct reflection of its beam from a mirror. A retro-reflective target shifts the beam polarization in a unique fashion, and only this returning polarization signature will trigger the detector. The disadvantage of this scheme is that it requires some modification of the robot's operating environment, although the tape strips are relatively unobtrusive.

Once you get the basic behavior primitives up and running, buy a cheap electronic compass and see what performance improvements result from this additional navigational information. The task of exiting a room, for example, is greatly simplified if you can positively identify the correct wall to scan for an opening. Implement a dead-reckoning capability and improve your algorithms even more. Incorporate time-out or exit conditionals for your behavior primitives to preclude getting caught in an endless loop when something goes wrong. Augment your navigational code with trap recovery routines that take over if the robot gets boxed into a repetitive cycle that leads nowhere. Add exception handlers to deal with opposing doorways, a door situated at the very end of a hallway, or a room with two doors.

6.3 Herbert

In the late eighties, as part of his PhD work at the MIT AI Lab, Jon Connell implemented a similar but much improved navigation scheme on the mobile robot *Herbert*. I had met Jon some years before when Anita brought him by my basement workshop in Springfield, VA, along with fellow grad student Peter Ning, who did much of the wiring and construction of *Herbert*. This was before Jon had started his PhD pursuit, and I was by then (i.e., 1984 or so) deeply involved in the development of *ROBART II*, so we did not talk at all about the hallway navigation scheme of *ROBART I*. It was at Anita's suggestion during the preparation of this manuscript in 2005 that I purchased and read a copy of Jon's book, *Minimalist Mobile Robotics*, which describes his research on *Herbert* (Connell, 1990). In retrospect, this chance disconnect was most unfortunate, given the many similarities in our respective navigational approaches.

Herbert was built upon an omnidirectional platform manufactured by Real World Interface, then located in Sudbury, MA, and featured a loosely coupled 24-processor computer architecture (Brooks, 1999). Its job was to roam the corridors at MIT and pick up empty soda cans off desks and tabletops with an onboard arm/manipulator (Connell, 1988), then transport the cans back to a collection area at the starting point. But most importantly from my perspective, the robot performed this navigational feat without relying upon a world model. In fact, the onboard computer architecture did not retain a state representation of any type, no matter how simplistic, for more than a few seconds. Seems quite the navigational challenge, to randomly explore a fairly complex indoor structure and then return to the starting point without remembering anything. Needless to say, I was quite intrigued!

For situational awareness, *Herbert* was equipped with a staring array of 30 short-range (i.e., 1 to 2 feet) near-infrared proximity sensors in two stacked rings as shown in Figure 6-9, which were very similar to the short-range proximity sensors I had designed for *ROBART I.* The sensors were evenly spaced around the robot's circumference to yield 16 discrete zones of coverage, although there were some intervening gaps due to beam misalignment and a fairly narrow field of view (Connell, 1990). There were only 14 sensors in the upper ring, on account of structural interference from the top-mounted manipulator, and these were logically ORed with their respective positions in the lower ring. In essence, *Herbert's* staring-array sensor coverage was identical with *ROBART's* 16-zone pie-shaped representation discussed in the last section, only spread over a full 360 (versus just the forward 180) degrees. This additional rear-area sensor coverage turned out to be quite useful, as we shall soon see.

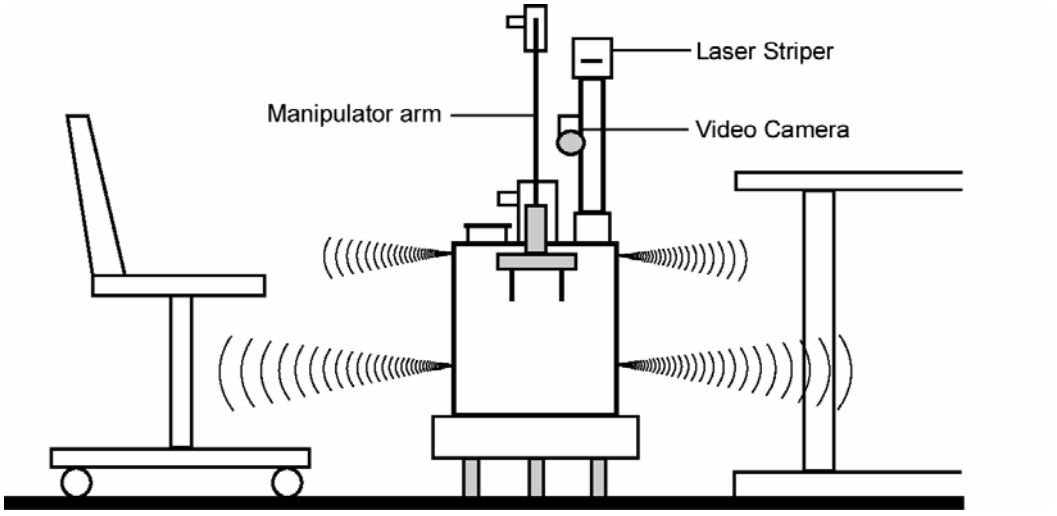


Figure 6-9. Two rings of near-infrared proximity sensors provide 360-degree spatial sampling in the vicinity of the MIT robot *Herbert* (adapted from Connell, 1990). The upper ring of sensors could detect out to 12 inches, while the lower ring had a maximum range of 24 inches.

6.3.1 Tactical Navigation

Connell uses the terminology *tactical navigation* in reference to the low-level behaviors that allowed *Herbert* to follow various intrinsic paths defined by local features of the environment. The first advantage offered by Connell's approach over *hallway navigation* was the added ability to traverse large open areas by *wall hugging* the perimeter, even with substantial wall-surface discontinuities due to support columns, furniture, appliances, etc. This improvement was achieved by arbitrating *two active* behaviors, one seeking to avoid and the other seeking to hug. This differed somewhat from the scheme I had implemented, which in contrast could best be described as *active avoidance* overriding *passive hugging*. As seen in Figure 6-10, *Herbert* would actively alter course to keep sight of the corner, whereas *ROBART I* would meander out into the open area, with a slight passive drift back to the right.

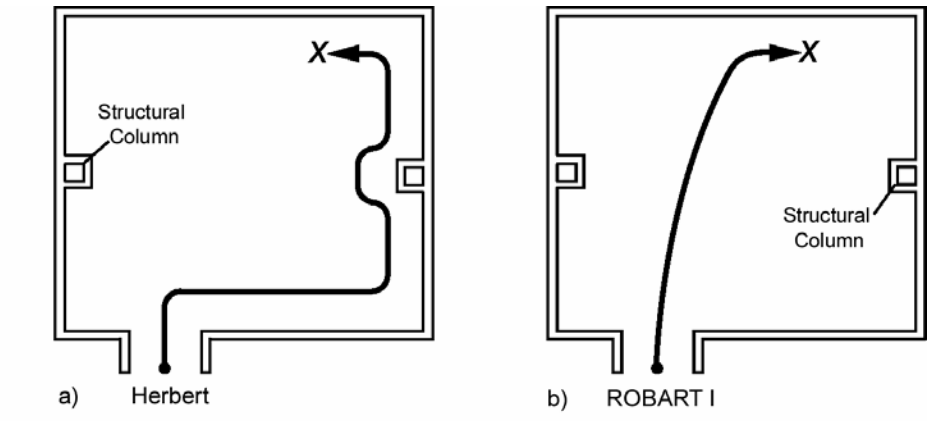


Figure 6-10. Upon entering a large room: **a)** *Herbert* turns to follow the wall; **b)** *ROBART I* continues moving forward with a passive drift. Both reach the same spot *X* on the opposite side of the room, but *Herbert* is better able to retrace its path.

The *Veer* behavior on *Herbert* (Figure 6-11b) used the four forward-most sensors to steer away from obstacles in similar fashion to the reflexive collision-avoidance sensors that supported the *Wander* behavior on *ROBART I*. Additionally, *Herbert* had a simultaneous requirement to always keep something in range of at least one proximity sensor in its staring array of 30 (much like ancient mariners traveled up and down the coast in sight of land, until such time as celestial navigation broadened their options). This *Hug* behavior only kicked in if there were no detected objects in front of the robot, and when it did, it turned the robot towards the nearest target to the sides or rear (Figure 6-11c). If none of the sensors detected a nearby presence, the robot would stop and back up to regain contact. In this one regard, Connell's scheme was very similar to the earlier wall-following strategy I unfortunately had abandoned, which relied upon one sensor to start the avoidance maneuver and another to stop it. I apparently had not used enough sensors.

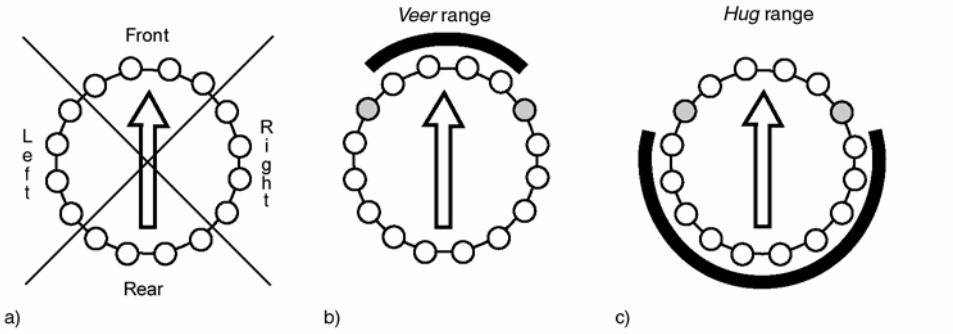


Figure 6-11. a) The sixteen proximity-sensor zones were divided into four quadrants of four zones each; **b) t**he *Veer* behavior avoids obstacles in the forward quadrant of four sensors; **c)** the *Hug* behavior turns toward the nearest object if *Veer* is not invoked. These opposing behaviors collectively worked to keep objects at a constant azimuth indicated by the gray sensors (adapted from Connell, 1990).

The combined effect of the *Veer* and *Hug* behaviors is perhaps better illustrated in Figure 6-12. If the robot was too far away, *Hug* caused it to alter course slightly toward the wall, whereas if it was too near, more sensors would detect, including one in the front quadrant (see again Figure 6-11a), which triggered *Veer* to turn away.

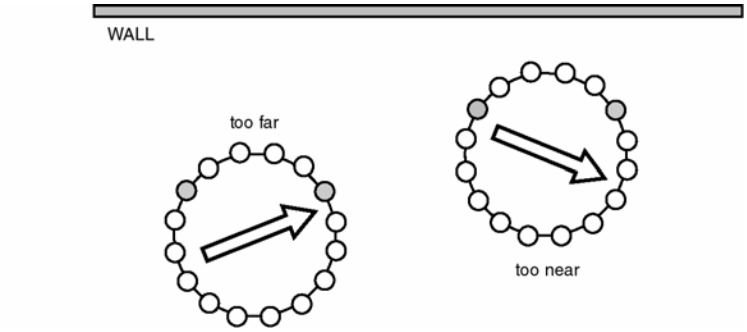


Figure 6-12. The combined effect of the *Veer* and *Hug* behaviors caused *Herbert* to alter course to keep the leading edge of the sensor pattern at an optimal angle (adapted from Connell, 1990).

6.3.2 Strategic Navigation

Connell's *strategic navigation* component was tasked with selecting the appropriate direction of travel when *tactical navigation* arrived at a path branch, both in outgoing exploratory mode as well as when returning to *Home*. His solution was elegantly simple: explore outward in a generally northerly direction, and always take the southern option when returning home. (Not unlike a lot of my high-school classmates, who left South Carolina for college in the late sixties.) Thus, *Herbert* was hard-coded with a single piece of information indicating which way to turn to get back to its origin, but there was no *a priori* model or topological map. Connell used doorway detection to recognize points where a branch was required, and added an electronic compass to identify the heading options by sensing the earth's magnetic field. In this fashion, the robot's *world model* of its surroundings was the world itself, both spatially and magnetically speaking.

There is an intriguing parallel here to Alva Noe's more recent challenge to the computational-school theory of "pure vision" (in the terminology of Churchland et al., 1994). Proponents of this traditional approach (i.e., Marr, 1982) think of vision as "generating a detailed internal representation of the visual world on the basis of information available to the retina alone" (Noe, 2004). Noe poses the following question:

> "An active approach to perception raises a more significant concern. If the animal is present *in* the world, with access to environmental detail by movements – that is, if it is active, embodied, environmentally situated – then why does it need to go to the trouble of producing internal representations good enough to enable it, so to speak, to act as if the world were not immediately present?"

Indeed, why go to great lengths to build an elaborate model if you are situationally immersed in the real thing? As simple as that sounds, there were some minor real-world wrinkles with *Herbert's* minimalist approach, for such is the proverbial nature of autonomous mobile robots. One obvious tradeoff was efficiency, but in some applications the amount of time required to move from one point to another is not that important, particularly if some useful secondary task (such as surveillance) can be performed enroute. In addition, the fact that *Herbert* only explored in one general direction (i.e., north) was for the most part inconsequential if the *Home* position was situated at the other (i.e., southern) extremity of the building layout.

The biggest problem in *Herbert's* navigational strategy was that it somewhat restricted the ability to explore, for the requirement to always pick a southerly branch when returning meant the robot must not enter center types of doors when outward-bound. This peculiarity resulted in some regions of the surrounding environment never being explored at all, which was an acceptable tradeoff, considering *Herbert* did not require any *a priori* description of room connectivity, nor was it constrained to operate only along hallways. Without getting into too detailed a discussion, the outgoing "explore" algorithm would not enter *north-south* doors aligned the same as the *Home* door as shown in Figure 6-13.

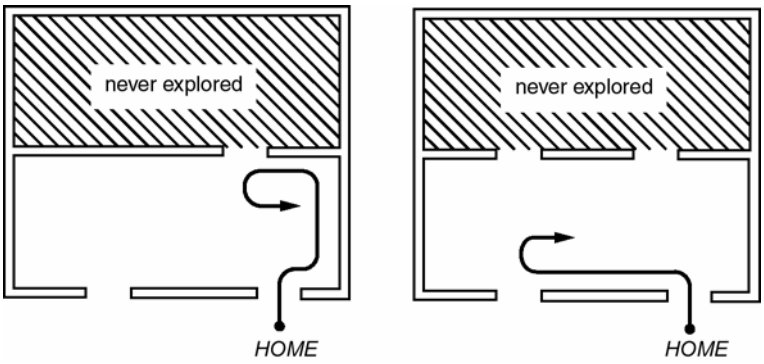


Figure 6-13. In outgoing "explore" mode, *Herbert* would not enter doorway orientations identical to the *Home* door, and would instead switch modes to retrace its path and return home (adapted from Connell, 1990).

If the robot encountered such a doorway while traveling east or west, it would terminate outbound exploration, turn around, and retrace its path back to *Home*. Another way to describe this discriminatory preference is that *Herbert* would only enter doors that were orthogonal to the northbound away-from-home heading, such as office doors along a *north-south* hallway corridor. While exploring, the robot essentially had to be traveling *north* (i.e., away from *Home*) when it encountered any door (Figure 6-14), so *south* would be the correct turn direction when it retraced its route back through that door on the return leg. For a more in-depth discussion of these behavior rules, see chapter 5 in *Minimalist Mobile Robotics* (Connell, 1990).

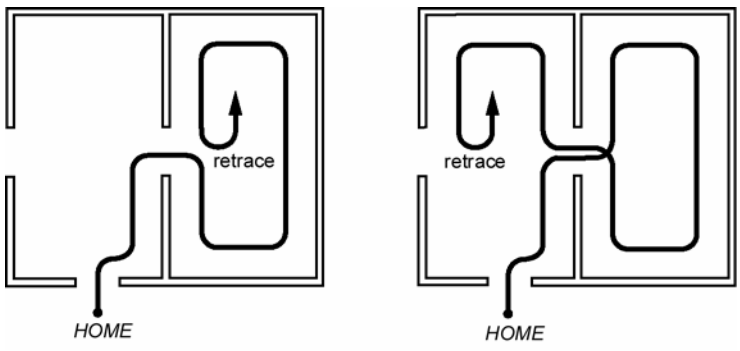


Figure 6-14. The "retrace" return path is also triggered by the first discovery of an east–west doorway while on a southerly heading (adapted from Connell, 1990).

How did the software know which way *Herbert* was traveling in order to make these decisions? The robot employed a Zemco digital fluxgate compass that had been modified to increase its update rate to about 10 Hertz, which reportedly had a worst case deviation of about 45 degrees in the presence of high iron content. Obviously this level of accuracy is not suitable for closed-loop heading control, but it was good enough to classify a path option in terms of its cardinal heading (Connell, 1990):

> "Furthermore, because intersections tend to be orthogonal, we do not need incredible angular resolution to clearly distinguish which path to take. All that matters is that the compass be locally consistent over some small area in front of the door, and that it be temporally consistent so that it points in the same direction when the robot retraces its path."

As it turns out, I had started hacking an analog version of the Zemco compass (Figure 6-15) on *ROBART II* in 1987 (before the digital version had been introduced), and come to a similar conclusion, although for a slightly different application involving an absolute world model. We had incorporated a sonar-based wall-following algorithm that performed a linear-regression line fit to a series of range-offset values, thereby establishing the angle of the wall surface relative to the robot's nearly parallel trajectory (Everett et al., 1988; 1989). The idea was to broadly classify a wall as either *east–west* or *north–south*, based on the compass reading, and then use the line-fit solution to precisely quantify the actual heading in building coordinates. As low-cost compass technology steadily improved over the next few years to yield better accuracy and resolution, this hybrid compass/sonar approach evolved into the *Orthogonal Navigation* capability we later employed on *ROBART III* (Ciccimaro et al., 1998).

Figure 6-15. A Zemco *DE 700* analog compass was used on *ROBART II* to determine the cardinal heading of a wall-following path segment.

Once again, it is a shame we were not able to exploit the obvious synergy that a mutual collaboration would have offered, had we only known.

6.4 Dervish

A more recent "classical-AI" approach to hallway navigation is seen in the robot *Dervish*, conceived by a team from the Computer Science Department at Stanford University to operate in reasonably structured office corridors (Nourbakhsh, 1998). *Dervish* was specifically designed as a contender for the AIII National Robot Contest, and won the "Office Delivery" event in 1994. The mechanical construction of *Dervish* was based upon a modified *NOMAD 100* robot, stripped of its original near-infrared

range sensors and tactile bumpers and re-outfitted with a custom sonar system as shown in Figure 6-16. As was the case with *ROBART I*, there were no rear-facing sonars, a design decision again based on the biological precedent observed in humans (Nourbakhsh, 1995):

> "Our basic philosophy was that humans clearly don't need eyes in the back of their head to navigate office buildings; so, neither should our robot."

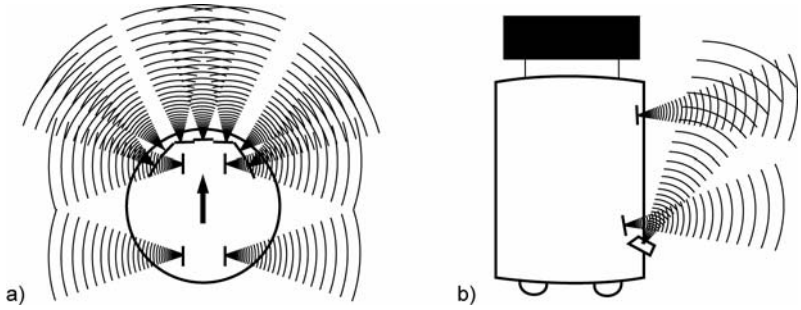


Figure 6-16. The custom sonar system on *Dervish* consisted of: **a)** three "clusters" of redundant Polaroid® sensors covering the forward direction of travel, with two fore-and-aft sensors facing outward on either side; **b)** an upward-angled sonar on the front (adapted from Nourbakhsh, 1998).

6.4.1 Collision Avoidance

The Stanford team opted for a case-specific, rule-based collision-avoidance strategy over other alternatives (i.e., weighted sum, potential field, vector histogram), primarily to decouple the various avoidance behaviors for fine-tuning and debugging. In other words, they tailored specific reactive routines on a case-by-case basis, depending on the conditions encountered, as defined by sets of sonar values grouped into *categories*. One such categorical distinction, for example, would be all cases in which the upward-pointing sonar sensors returned range values indicating close proximity of a reflective surface, versus longer ranges signifying no such blockage. In the structured hallway scenarios addressed, Nourbakhsh (1998) reports this case-based collision-avoidance scheme proved very reliable: "During the entire 1994 Robot Contest, *Dervish* never collided with anything."

6.4.2 Map Representation

Dervish's navigation scheme was specifically driven by the entry rules of the National Robot Contest, which generally described the type of operating environment, as well as the level of *a priori* information to be provided. The robot would be given a topological description of room connectivity, but no metrics describing geometric distance or angle. Starting in a known room, it would then be commanded to go to a specific room destination, despite any interfering obstacles strategically placed along the way. Accordingly, Dervish generated a discretized *state-map representation* as seen in Figure 6-17, with each *state* described by a node on the topological map (shown as a number) or a hallway segment between nodes (shown as two numbers separated by a dash).

Each of these abstract states in turn represented a number of possible robot positions within the associated geometric boundaries of that path segment or node. Precisely

where the robot was located was not important, only the fact that it was somewhere within that particular state. The total lack of geometric metrics in the original topographical site description meant it was impossible to directly correlate state occupancy with dead-reckoning estimates. This positional uncertainty was addressed through the concept of *state sets*, which represented all *possible states* in which the robot conceivably could be located at any given time (Nourbakhsh, 1998).

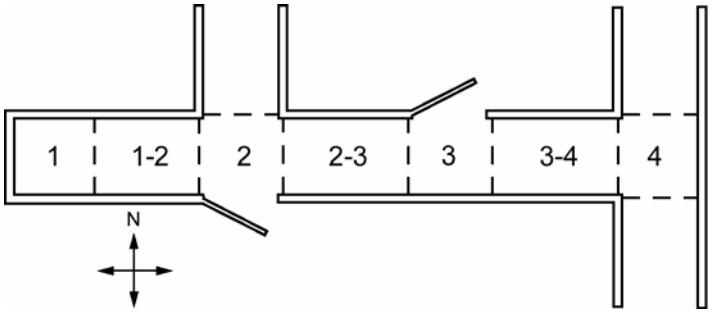


Figure 6-17. A *state-map representation* of nodes and path sections between nodes, generated from the furnished topological description of room/hallway connectivity (adapted from Nourbakhsh, 1998).

6.4.3 Feature Recognition

The problem, of course, was ascertaining the robot's *actual state* (i.e., the specific topological node or path segment occupied at that particular point in time) from the various possibilities contained in the *state set*. To address this need, a sonar-based feature detector, tuned to an office environment, allowed *Dervish* to distinguish walls, closed doors, open doors, and intersecting hallways. The robot had to stay aligned with the hallway axis for the feature detector to work, but could momentarily deflect as required to skirt obstacles blocking the path.

Proper alignment was determined by comparing range values from the two side-facing sonars mounted towards the front and rear on each side (see again Figure 6-17). On any given side, in other words, near-identical range readings from these two sensors implied beam orthogonality with respect to the wall surface, a condition that minimized range errors due to *specular reflection* (Everett, 1995a; 1995b). This approach also reduced the earlier position-uncertainty problem to one of longitudinal translation only (i.e., no uncertainty in lateral position or orientation), which greatly simplified matters.

The *feature detector* essentially compared the sonar-measured width of the hallway with the expected width, and classified the results according to predetermined rules (Nourbakhsh, 1998). If the measured range was much greater than expected (i.e., 60 inches or more), for example, the robot was passing an open door or hallway intersection. Conversely, if the difference was only 3 to 7 inches, the robot may have been passing a closed door. Any perceived discontinuity could be identified with the appropriate side of the hallway by looking at the history of measured range values for each side-facing sonar pair, before and after state transition.

Furthermore, the width of an observed opening could be derived from the robot's dead-reckoning calculations, thus discriminating between hallways and doors. In this

fashion, *Dervish* could reliably extract relevant state descriptors, termed *percept-pairs*, using fairly noisy low-resolution sonar sensors. As could reasonably be expected, detection of closed doors was less than 70-percent reliable, due to the poor range and angular resolution of ultrasonic sonar operating in air. Additional problems were encountered in the presence of random obstacles.

6.4.4 High-Level Reasoning

To compensate for such low-level sensor inaccuracies, the Stanford team incorporated a high-level reasoning system based on the concept of *state set progression*. When the robot detected a *percept-pair* while moving down the hall, it would update the *state set* accordingly by "progressing" each *state*, based on the observed *percept-pair* and its associated direction of travel. Nourbakhsh (1998) defines such *progression* of a state as "removing it from the state set and replacing it with all possible subsequent states." This is perhaps better understood by considering the following example, using the topological state-map representation presented earlier in Figure 6-17.

The robot is traveling west with the *state set* of possibilities {2-3, 1-2}. If the feature-detector then reported a *hallway* on the right and an *open door* on the left, this *percept-pair* uniquely establishes the robot to be at *node 2*. So, the previously possible state 2-3 progresses to {2}, while 1-2 progresses to the empty set, since the current *percept-pair* is impossible if the world state was 1-2. This simplistic example assumes perfect sensors, however, which is never the case with sonar, particularly with regard to closed-door detection as previously discussed. Accordingly, this *state set progression* approach was augmented with certainty factors that described the *likelihood of detection* for each possible *percept-pair*, the likelihood of mistaking a hallway intersection for an open door, for example, being only 10 percent.

Using an *assumptive planning* strategy, *Dervish* assumed its current state to be the most likely candidate in the *state set*, and generated a sequential solution leading to the desired goal state. While executing that plan, the robot "progressed" its *state set* based on run-time output of the feature detector, until the current state became the goal, or was found inconsistent with the intended path. In the latter case, the system had to stop and replan, having either overshot the turn into the specified room destination (i.e., the goal), or else localized itself in a completely different hallway (Nourbakhsh, 1998).

6.5 References

Anderson, T.L., and Donath, M., "Synthesis of Reflexive Behavior for a Mobile Robot Based Upon a Stimulus-Response Paradigm," SPIE Mobile Robots III, Vol. 1007, W. Wolfe, Ed., Cambridge, MA, pp. 198-211, November, 1988.

Arkin, R.C., "Motor-Schema-Based Navigation for a Mobile Robot: An Approach to Programming by Behavior," IEEE International Conference on Robotics and Automation, Raleigh, NC, 1987.

Arkin, R.C., "Behavior-Based Robot Navigation for Extended Domains," *Adaptive Behavior*, Vol. 1, No. 2, MIT, Cambridge, MA, pp. 201-225, 1992.

Banner, *Photoelectric Controls*, Product Catalog, Banner Engineering Corp., Minneapolis, MN, 1993a.

Banner, *Handbook of Photoelectric Sensing*, Banner Engineering Corp., Minneapolis, MN, 1993b.

Borenstein, J., and Koren, Y., "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," IEEE International Conference on Robotics and Automation, Vol. CH2876-1, Cincinnati, OH, pp. 572-577, May, 1990a.

Borenstein, J., and Koren, Y., "Real-Time Map Building for Fast Mobile Robot Obstacle Avoidance," SPIE Vol. 1388, Mobile Robots V, Boston, MA, November, 1990b.

Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-20, 1986.

Churchland, P.S., Ramachandran, V.S., and Segnowsky, T.J., "A Critique of Pure Vision," in *Large-Scale Neuronal Theories of the Brain*, C. Koch and J.L. Davis, Eds., MIT Press, Cambridge, MA, pp. 23-60, 1994.

Ciccimaro, D.A., Everett, H.R., Gilbreath, G.A., and Tran, T.T., "An Automated Security Response Robot," SPIE Mobile Robots XIII, Boston, MA, 1-6 November, 1998.

Connell, J.H., *A Behavior-Based Arm Controller*, Memo 1025, MIT AI Lab, Cambridge, MA, 1988.

Connell, J.H., *A Colony Architecture for an Artificial Creature*, Technical Report 1151, MIT AI Lab, Cambridge, MA, 1989.

Connell, J.H., *Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature*, ISBN 0-12-185230-X, Academic Press, San Diego, CA, 1990.

Everett, H.R., "A Computer Controlled Sentry Robot," *Robotics Age*, March/April, 1982a.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot*, Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982b.

Everett, H.R., Gilbreath, G.A., and Bianchini, G.L., *ROBART II: An Autonomous Sentry Robot*, NOSC Technical Document 1230, Naval Ocean Systems Center, San Diego, CA, March, 1988.

Everett, H.R., and Gilbreath, G.A., *ROBART II: A Robotic Security Testbed*, NOSC Technical Document 1450, Naval Ocean Systems Center*, San Diego, CA, January, 1989.

Everett, H.R., "Understanding Ultrasonic Ranging Sensors," *The Robotics Practitioner*, Vol. 1, No. 4, Fall, 1995a.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, AK Peters, Ltd, Wellesley, MA, 1995b.

Everett, H.R., "Autonomous Navigation on a Shoestring Budget," *The Robotics Practitioner*, pp. 15-23, Winter, 1996.

Everett, H.R., Irie, R., Pacis, E., Kogut, G., and Farrington, N., "Towards a Warfighter's Associate: Eliminating the Operator Control Unit," Proceedings, SPIE Unmanned Ground Vehicle Technology VI, Defense and Security, Orlando, FL, 12-16 April, 2004.

Flynn, A.M., Brooks, R.A., and Tavrow, L.S., "Twilight Zones and Cornerstones: A Gnat Robot Double Feature," MIT AI Laboratory Memo 1126, July, 1989.

* Now Space and Naval Warfare Systems Center San Diego.

Hollar, S., Flynn, A., Bellew, C., and Pister, K.S.J., "Solar Powered 10-mg Silicon Robot," MEMS 2003, Kyoto, Japan, January 19-23, 2003.

Jones, J., and Flynn, A.M., *Mobile Robots: Inspiration to Implementation*, ISBN 1-56881-011-3, AK Peters, Ltd., Wellesley, MA, pp. 106-111, 1993.

Marr, D., *Vision*, W.H. Freeman and Sons, New York, NY, 1982.

McManis, C., "Turning Toys into Tools," *Circuit Cellar INK*, Issue #63, pp. 24-35, October, 1995.

Nourbakhsh, Illah, "The Sonars of Dervish," *The Robotics Practitioner*, Vol. 1, No. 4, Fall, 1995.

Nourbakhsh, Illah, "Dervish: An Office-Navigating Robot," in *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, Kortenkamp, Bonasso, Murphy, Eds., ISBN 0-262-61137-6, AAAI Press, Menlo Park, MIT Press, Cambridge, MA, pp. 73-90, 1998.

7
Summary

"It has yet to be proven that intelligence has any survival value."

Arthur C. Clarke

ROBART I was built on an extremely limited budget and its performance was significantly constrained as a consequence. It moved very slowly with way too big a turn radius, lacked sufficient sensors for full situational awareness, and had but a single eight-bit microprocessor to control its many actions. Nevertheless, I learned a tremendous amount during my 2 years of experimenting with this first prototype, partly because I started with absolutely no knowledge of computers, and not much more about robots. This closing section reviews some of my thoughts at the time with regard to an improved design for *ROBART II*.

7.1 Second-Generation Concept

If I had to summarize my perspective at this point in a nutshell, it would be that true autonomy on a mobile robot was a matter of providing appropriate sensors to perceive the surrounding environment, sufficient computational resources to process this information, and optimized control and actuation schemes to then interact accordingly. All things considered, the development of this first-generation prototype had resulted in a fairly sophisticated autonomous robot, but results clearly indicated the need for future versions to use additional sensors as well as computing power to enable more sophisticated behaviors. As you may recall from section 4, *ROBART I* had a mechanism for the ordered sequencing of behavior primitives, but no run-time ability for generating the sequence. Accordingly, one of the key upgrades under consideration was some high-level task-decomposition and planning functionality to address this deficiency.

The proposed block diagram for this multi-processor system as initially envisioned in 1982 is shown in Figure 7-1. Note that there was still no provision for any remote host computer or operator control unit, in part due to the fact that there were still no inexpensive readily available RF modems for serial communications. Some of the more relevant issues with regard to the envisioned hardware and software designs are discussed in the following subsections, starting at the bottom and working upwards.

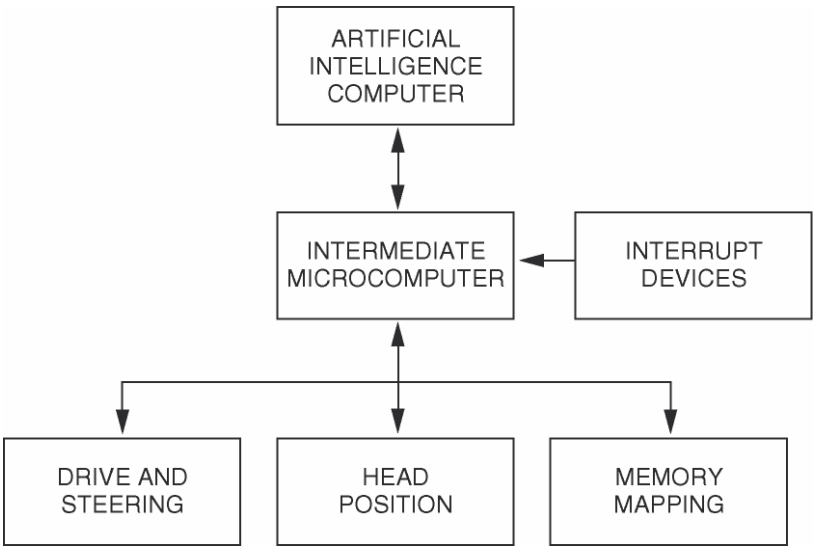


Figure 7-1. Proposed block diagram for the expanded next-generation computer architecture.

7.1.1 Head Pan Axis Controller

One direct result of my thesis work was exposure to software options versus an over reliance on dedicated hardware. For example, it was clear the head controller function could be better performed by a microprocessor as opposed to the hard-wired logic circuitry used on *ROBART I.* This approach would allow an eight-bit A/D conversion of the head position, providing better accuracy for processing information from head-mounted sensors. In addition, precise position-error feedback would make velocity control possible, resulting in a much more robust electromechanical response. A computerized tracking algorithm could keep the head centered on the beacon far more accurately than the original bang-bang solution I had devised. Now that I regarded inexpensive single-board microprocessors as just another building block in the overall design, there seemed no end to the possibilities!

7.1.2 Drive Controller

Significant performance gains could similarly be realized with a computerized drive controller. For improved maneuverability, I had been contemplating a drive-and-steering configuration with two independently driven wheels to allow the platform to turn in place. This arrangement would eliminate the need for a steering motor and its associated position-sensing potentiometer and A/D converter. Motor direction as well as individual motor speed could be precisely controlled through pulse-width-modulation, allowing an almost infinite range of turning radii for use in navigational and docking routines. As added incentive, trying to twist *ROBART I's* drive wheel in place while otherwise stationary on a carpeted surface had proved problematic, often skewing the carpet itself and leaving an unsightly scuff mark.

But more importantly, such wheel slippage contributed to dead-reckoning inaccuracies, somewhat of a moot point on *ROBART I,* which had had no dead-reckoning

capability of any sort. Accurate orientation/displacement/velocity information was a key goal for the second-generation system, however. A dedicated microprocessor running a proportional-integral-derivative (PID) control algorithm, I reasoned, could provide precise odometry (i.e., heading and displacement) information. And for efficiency's sake, the collision-avoidance strategy in its entirety could be run on this same processor.

7.1.3 World Modeling

I had come to the conclusion that near-infrared proximity detectors were better suited to the collision-avoidance application than ultrasonic sonar units for a number of reasons. The directional characteristics of near-infrared sensors make it possible to operate numerous devices in close proximity to each other without cross coupling, while their simplicity and low cost facilitated volumetric situational awareness. My bias towards optical versus acoustical sensors, however, was largely because I had experience with only one of each type, and in that limited comparison, the *LM1812* driving a *piezoelectric* transducer did not fare so well. The soon-to-be-introduced Polaroid® *electrostatic* sonar ranging system would turn the tables substantially, providing low-cost and reliable operation out to considerable distances (Figure 7-2). As an added bonus, these "ranging units" had the inherent benefit of reasonably accurate distance information, which the near-infrared proximity sensors lacked entirely.

Figure 7-2. Introduced in 1982, Polaroid's® ultrasonic ranging module would for the next decade become the sensor modality of choice for the mobile robotics research community. The more extensive evaluation kit shown above would later offer a range of transducers, but the original 1.5-inch sensor designed for camera autofocus remained the most popular (courtesy Polaroid Corporation®).

In addition to improved collision avoidance, it seemed obvious to me that such a reasonably accurate ranging capability could potentially be exploited to significantly augment the robot's odometry solution. Accordingly, I allocated another microprocessor to the task of repeatedly determining vehicle position from the perceived environment, a functionality now commonly known as "localization." This *Mapping Module* would maintain an updated memory map of the surroundings, specifically noting the locations of the battery recharging station, doorways, and other relevant items, in addition to

obstructions. I did not realize it at the time, but this was my first reference to the notion of *augmented virtuality* (Milgram and Kishino, 1994), which would become a key aspect of the *Warfighter's Associate Concept* later pursued on *ROBART III* (Everett et al., 2004). The development of cost-effective hardware to more precisely determine position and orientation remained a crucial means toward this end, but the Polaroid® ranging module was a very promising first step.

7.1.4 Intermediate Processor

The *Intermediate Processor* was intended to serve as an interface between the robot's many subsystems and some eventual goal-oriented AI program written in ADA or LISP. (Looking back on it now, I am not sure what triggered this obscure reference to ADA.) For lack of suitable low-cost alternatives, I selected the *SYM-1* single-board computer to again serve in this capacity. Assembly language programming seemed well suited to the needs of the individual dedicated controllers listed above, yielding fast and efficient code for the direct manipulation of control lines and/or the reading of sensory inputs. I envisioned programming the *Intermediate Processor* in assembly language as well, even though versions of BASIC and also FORTH were by then available for the *SYM*, because there would still be a lot of bit-intensive I/O lines connected at this level. My ignorance was once again in evidence, but one thing at a time. I learned a lot by starting at the very bottom.

7.2 Towards Artificial Intelligence

I must admit that I almost left all mention of this particular topic out of this book adaptation of my thesis, because in the intervening years I became somewhat wary of the term "artificial intelligence" as applied to robotics. What limited reference to the subject that does appear in my thesis was a direct result of a computer-science elective I took just before graduation. (I do not remember the professor's name.) The purpose of the reading course was for me (a mechanical engineering major) to get some exposure to this new and much-hyped field of research that seemed destined to play a major role in the future of intelligent robotics.

There was a decided disconnect, however, in that my instructor did not have any background in robotics, and I had no real background in computer science, much less artificial intelligence. He talked about knowledge representation, expert systems, and planners, then referred me to examples such as *MYCIN*, an expert system for medical diagnosis, and *STRIPS*, the *STandford Research Institute Problem Solver*, a popular planning algorithm developed in the seventies. I, on the other hand, primarily wanted to better understand how to make a truly intelligent robot. Tracking down information on pioneering AI efforts without the benefit of today's search engines crawling the World Wide Web was tedious and time-consuming. With only one academic quarter left, I was very short of time.

But I was quite intrigued about the concept of ultimately imparting human-like intelligence to one of my future robotic creations, just not at all clear as to how to proceed. For starters, what was the definition of ordinary intelligence as manifested in humans? Perhaps equally important, what was the appropriate definition of a robot?

Since neither of these terms had been pinned down in their own right to my satisfaction, their combined meaning in the terminology *intelligent robot* was even more intangible. Furthermore, I wondered, how exactly would one describe the difference between *intelligence* and *common sense*, and was any such distinction even applicable in the case of a robot? I have known some very intelligent people who clearly had no common sense, and vice versa.

The de facto standard for defining a truly intelligent computer had been proposed in 1950 by Alan M. Turing, a British logician and leading cryptanalyst who played a role in cracking the German Enigma cipher during World War II. Turing was a brilliant visionary and computer pioneer who made substantial contributions in what he initially termed "machine intelligence," thus ushering in the era of AI as we know it today. Way back in 1935, it had been Turing himself who first conceived the modern computer, so his thoughts on making one truly intelligent seemed relevant indeed. The *Turing Test*, as it came to be widely known, proposed an impartial human evaluator (actually, a series of evaluators) communicating remotely via a keyboard with another human, and also with a computer. If the evaluators were ultimately unable to determine which responder was which in an unbounded natural-language interrogation, then the computer must have exhibited sufficient intellect to have successfully emulated a human being, and thus could be considered truly intelligent.

I initially accepted the *Turing Test* as a perfectly adequate metric for comparison of machine intelligence to human intelligence, but after thinking about it further, I became less convinced. I had just built an autonomous security robot that could move about a semi-structured environment in a somewhat purposeful fashion, avoiding obstacles as it did so, recharging its own battery when needed, and voice outputting meaningful information related to its status in plain English. In short, it demonstrated what appeared to be intelligent behavior, and garnered some incredible publicity as a consequence. But as a machine, it was not by any stretch of the imagination intelligent in the true sense of the word, in that it had no conscious awareness of what it was doing or why. In short, *ROBART I* was merely executing preprogrammed responses to changing input conditions, oblivious even to the fact that it was doing anything at all.

The fundamental issue in my mind was whether intelligent behavior maps directly over into true intelligence. As discussed above, I had some serious reservations. Actors, I reasoned by way of example, were paid to essentially emulate other human beings, not at all unlike the ambitious computer in Turing's envisioned competition. Surely there had been actors blessed with only average intelligence playing the part of (and hence attempting to emulate) far more gifted and intelligent historical figures. Did success in this regard not make that actor appear intelligent in his or her role to the various members of the audience? But this outward appearance of intelligence was merely the projection of an image, made possible by the actor executing a pre-planned script intended to convey that very effect. Was this not a direct analogy to *ROBART I* executing pre-planned collision avoidance maneuvers that had been painstakingly optimized through months of trial and error on the part of the programmer?

Consider also the following example. I have on occasion stated that it is not nearly as important for the President of the United States (or any prominent leader, for that matter) to be highly intelligent as it is to be an effective and motivational leader. The one caveat I quickly added was that such an individual must establish and heavily rely upon a cabinet of experienced and intelligent advisors, in order to "make" and then subsequently implement the correct choices. Hypothetically speaking, let us just assume for the moment that a Hollywood movie star were to somehow get elected to our country's highest office. Since elections are influenced for the most part by popularity, it is certainly possible he or she won as a result of their projected public image, a skill refined over years of employment as a professional actor. Conceivably, such a person could avail themselves of appropriate coaching with regard to contemporary affairs of state, then routinely go on stage and project an air of intelligent authority on important matters of global interest. And as such, be afterwards regarded as seemingly intelligent and demonstratively effective!

In my mind, this was a good example of how the mere appearance of intelligent behavior is not sufficient grounds to establish that the performing entity is in fact intelligent. Yet even to this day the notion has persisted, that the degree of system intelligence is defined by the observed system behavior, for reasons articulated by Hawkins in his recent book entitled, aptly enough, *On Intelligence* (2002):

> "It is not difficult to understand why people – laymen and experts alike – have thought that behavior defines intelligence. For at least a couple of centuries people have likened the brain's abilities to clockworks, then pumps and pipes, then steam engines and, later, to computers. Decades of science fiction have been awash in AI ideas, from Isaac Asimov's laws of robotics to *Star Wars'* C3PO. The idea of intelligent machines *doing* things is engrained in our imagination. All machines, whether made by humans or imagined by humans, are designed to do something. We don't have machines that think, we have machines that do. Even as we observe our fellow humans, we focus on their behavior and not on their hidden thoughts. Therefore, it seems intuitively obvious that intelligent behavior should be the metric of an intelligent system."

The other concern I had with a robotic adaptation of the Turing Test was the imposed constraint of a natural-language interface. To me, natural-language understanding was but one piece of many that made up what I vaguely envisioned at the time to be human intelligence. Admittedly, an ability to speak and understand language was probably a large part of what computer scientists thought of as AI, but even there I did not feel it to be wholly representative. *ROBART I* could talk in a very rudimentary fashion, but it certainly was not capable of speech recognition, and by no means did it understand what it was saying. But I surmised a robot could be incapable of speech synthesis or recognition altogether and yet still be intelligent in other areas, like perhaps path planning, collision avoidance, or security assessment. But again, what exactly did that mean, to "be intelligent?"

I had a lot of questions concerning the relationship of language and intelligence. In trying to sort out how the human brain decomposes a problem into a series of tasks, or how it represented knowledge that might be used in solving such a problem, I tried to

single-step my mind through the process in an attempt to better understand what was happening. For some reason, the incremental steps seemed always couched in terms of a verbal description. In other words, as I mentally examined a particular step, I seemed to be describing it to myself in words. Was this simply an artifact of the analysis, or an inherent property of the actual representation in the brain itself? Was the ability to converse in a high-level language some type of prerequisite to intelligent thought? Surely this could not be the case. Or could it? Either way, it was a very intriguing concept that continued to occupy my thoughts, and I reminded myself more than once that I did not really know much about anything in the grand scheme of things.

The other thing I noticed during this protracted mental exercise was that I could only concentrate my thoughts on one item at a time, despite my perception of the human brain as a massively parallel computer able to multi-task several concurrent processes. There may have been a dozen steps involved in some arbitrary behavior, but I could only examine them individually. I somehow was aware of the multiplicity, but I could only focus in on a single entity, and had to then release it before examining the next. I saw this as very analogous to human vision, where we are vaguely conscious of all that falls within our peripheral field of view, but we can only focus in to examine in detail one portion of the scene at a time.

But getting back to the more fundamental issue, the mere appearance of intelligent behavior was to me insufficient in and of itself to establish the presence of actual intelligence. Even beyond this realization, what I was actually but ineffectually grasping for was some understanding of what exactly made up that which we collectively consider to be intelligence in general, and how such a thing could be embodied in a robot. I could easily envision myself making *ROBART I* (or better yet, its successor) more and more capable over the years, yet still altogether lacking in what I would call true intelligence.

So what was this *artificial intelligence* all about that had these computer scientists so excited, and how was it supposedly going to alter the course of subsequent events? I decided to ask some of the graduate students in the Computer Science Department to elaborate on the issue. "AI is nothing more than a bunch of if-then rules," one matter-of-factly explained. Another suggested AI was merely a function of the high-level language used by the programmer, almost implying that if one wrote code in LISP, the result was artificial intelligence. Even today, *LISP* and *Prolog* are regarded as "AI languages" since they are used almost exclusively for AI applications (Webopedia, 2004).

These and other attempted explanations I heard at the time sounded way too *artificial* and not all that *intelligent*. Probably because I lacked the background to fully appreciate the issues, the Naval Postgraduate School was not yet into the mainstream of AI or robotics research, and as a naval officer, I was about to be transferred far away to a new assignment on the east coast. In retrospect, the term "artificial intelligence" itself was probably somewhat of a misnomer, conjuring up images in my nascent mind of some magical rendering of human-like intelligence to an otherwise inanimate machine.

I could accept the fact that a bounded application such as an expert system was realistically possible using rather clever if-then rules, but I had a hard time imagining a computer that faithfully emulated human intelligence across the board. Besides, the example AI applications I had recently come to know required way too many computational resources for a battery-powered mobile robot, and I did not see an immediate need to replicate the full spectrum of human intelligence on a robot anyway. So I simply boxed in a rather vague square as a placeholder in my proposed hierarchy for some "Artificial Intelligence Computer," with the intent of one day revisiting the subject with a better frame of reference. Almost a quarter century later, I would still be looking for answers.

7.3 Enter ROBART II

In the meantime, I decided to continue my bottom-up development approach, with a particular focus on reactive collision-avoidance strategies and sensor-based navigational referencing. *ROBART I* had been intended as a very basic demonstration of technical feasibility, and was built on an extremely limited budget, using oversimplified approaches. The system maintained no absolute model of its environment, had no idea of its X-Y location or heading, and performed no dead-reckoning calculations of any type. Yet despite these limitations, the robot could operate for days at a time, seek out human presence, avoid obstacles in a dynamic environment, move from room to room in somewhat purposeful fashion, even recharge its own battery. My rationale assumed that if the concept of an autonomous security robot could be successfully demonstrated under such primitive conditions as these, a reasonable extrapolation would show promise indeed for a more sophisticated second-generation version (Figure 7-3).

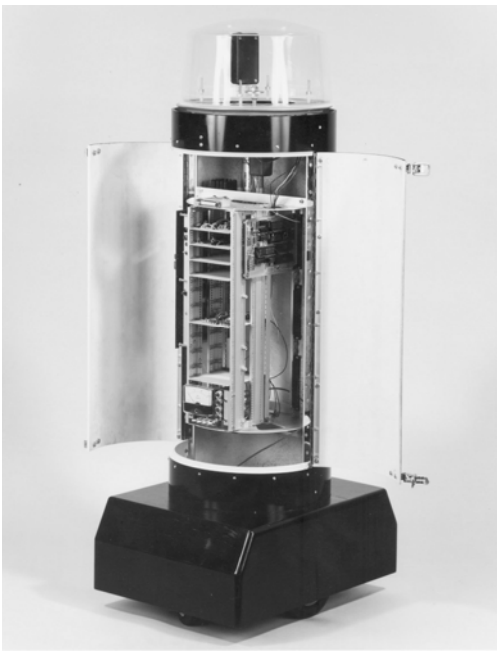
a)  **b)** 

Figure 7-3. a) Early stages of *ROBART II* in Monterey (circa 1982), with the computer card cage shown at right; **b)** rear view showing card-cage installation and clear plastic dome for head. The 12-volt air compressor and accumulator would have to wait for installation on *ROBART III* some 10 years later.

The body was constructed from a scrap piece of 13-inch plastic irrigation pipe. Strong, lightweight, and very easy to work, this material was ideal for prototyping. To fabricate the large access doors for the computer card-cage bay in the back (Figure 7-3b), I simply marked the desired openings, made the cuts with my radial arm saw, then reattached the two removed sections with piano hinges. I was also kicking around the idea of using a 12-volt air compressor to power pneumatically actuated manipulators, so I fabricated an air bottle from some smaller sections of pipe to serve as the accumulator. I eventually nixed the manipulator idea, in that I couldn't see any realistic application for manipulators on an autonomous robot without a vision system of some type, and that prospect certainly did not look promising for the near term. I would resurrect the air compressor concept years later in 1992 to power the pneumatic dart gun on *ROBART III*.

Since I had finished and presented my thesis a quarter early, I had some spare time to experiment with some new technologies. I ordered one of the Polaroid® ultrasonic ranging kits which had just come on the market, and a new 6502-based microcontroller (Figure 7-4) from R. J. Brachman, Inc., that looked like it would be ideal for the dedicated controllers below the *SYM* in the proposed hierarchy. In playing around with these components I decided to add another low-level processor to serve as a sonar controller, timing the echo return to yield range. (My thesis was already signed off and in printing, however, so it was too late to modify the block diagram.) I also decided that I would install at least six sonar units on *ROBART II*, so an additional function of this new processor would be to sequentially multiplex the hardware accordingly.

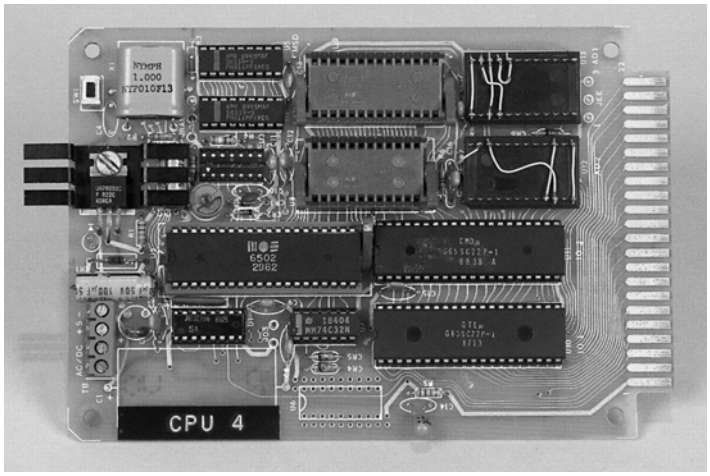


Figure 7-4. The *MMC-02* microcomputer featured two *6522 VIA* ports for a total of 40 external I/O connections, while the all-CMOS design substantially reduced power consumption.

7.4 Moving on to DC

Sometime during the spring of 1982 I gave an invited presentation on *ROBART I* to the Salinas/Monterey User's Group (SMUG) of *Apple* computer aficionados, the connection being that the 6502 microprocessor used in the *SYM-1* was also the heart of the *Apple*. Shortly afterwards I was contacted by the *Monterey Peninsula Herald* for a feature article that ran on 14 April 1982, portions of which were picked up by United Press International out of San Francisco for worldwide distribution the next day. After that, "all hell broke loose." The Naval Postgraduate School assigned a Lieutenant

Commander from their Public Affairs Office to sit at my house for about 4 days to answer the phone, which literally rang again as fast as you hung it up. I did three local television shoots and one national news segment with *ABC World News Tonight*, which flew a film crew into Monterey, plus over 30 live radio interviews with various talk show hosts and interested DJs. Back in those days, an autonomous security robot that could talk and patrol a home environment with no human intervention created quite a sensation.

One positive result of all this unanticipated (and ultimately annoying) publicity was that it greatly facilitated the efforts of CAPT G. R. Garritson, the Naval Engineering Curriculum Officer, in getting me reassigned to a specially created robotics billet at the Naval Sea Systems Command (NAVSEA). Upon completion of my studies, I was transferred to the Washington, DC area, where I initially served as Special Assistant for Robotics to COMNAVSEA, VADM Earl B. Fowler, and later became Director of the NAVSEA Office of Robotics and Autonomous Systems (SEA 90G). This new duty station was a huge turning point in my Navy career, as from then on, I was the designated point-of-contact in this new and exciting field. On the downside, relocation to the east coast seemed to start the media frenzy all over again.

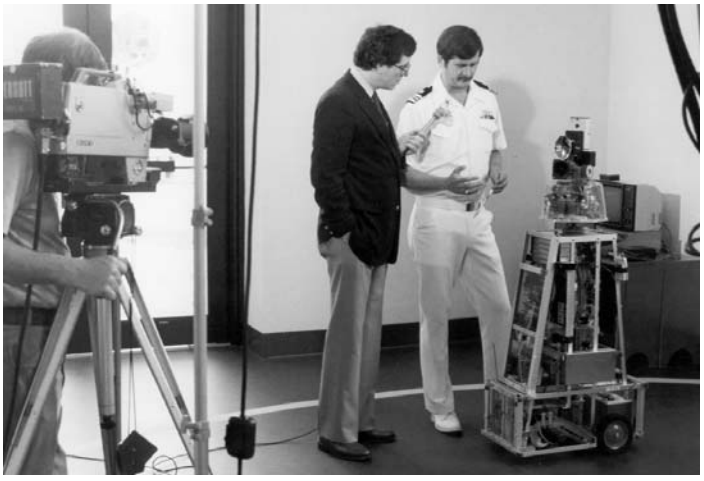


Figure 7-5. Filming *ROBART I* in action with a local news team at the Naval Surface Weapons Center, White Oak, MD, after relocation to the east coast.

Since my robotics oversight position at NAVSEA had been newly created by decree, there was no established organization or procedures, which gave me some latitude in setting things up to my own liking. I wasted no time in arranging for a good part of my schedule to be spent in hands-on pursuit of technical issues at some of the Navy labs in the DC area, specifically the Naval Surface Weapons Center (NSWC), the Naval Ship Research and Development Center (NSRDC), and the Naval Research Lab (NRL). I sanctioned this arrangement by defining part of my charter to be the identification of common technological hurdles impeding progress in multiple NAVSEA robotic interest areas, and then personally supervising a distributed team of engineers at these laboratories in addressing said deficiencies. The range of envisioned NAVSEA applications at that time included: (1) ship and weapon systems manufacturing, (2) ship maintenance and repair, and (3) operations (Everett, 1984).

Since my Navy job was mostly programmatic responsibilities, I had to continue the development of *ROBART II* on my own time, and using my own resources. But it was precisely these restrictions that fostered a synergistic working relationship with Anita Flynn, the MIT co-op student I had met out at the Naval Surface Weapons Center in White Oak, MD, now guardian of *ROBART I* (see section 6). Anita and I quickly became good friends, spending many a late night and lots of weekends hacking software in my basement workshop in Springfield, VA. It was through this close association that I came to meet Dr. Mike Brady of Oxford University, then doing an exchange assignment at the MIT AI Lab, and later Professor Rodney Brooks, who became Head of the AI Lab in 1997.

To facilitate the *ROBART I's* new role as a research testbed, we made a few minor upgrades to various subsystems to improve reliability and extend the operational lifetime, as the prototype hardware was never intended to survive much past my final thesis demonstration. (I give Anita full credit for talking me out of cannibalizing *ROBART I* for parts, which is the only reason this somewhat historical robot is still around today.) One of the first such enhancements was a rugged and much improved cylindrical recharging station. This new design, which was also compatible with *ROBART II*, featured an enclosed aluminum housing, short-circuit protection, and a more reliable RF control link to activate the beacon.

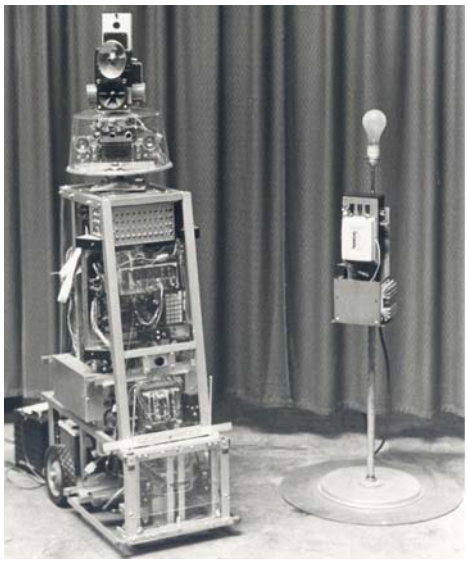
a)  **b)** 

Figure 7-6. a) The second-generation recharging station is shown here next to *ROBART I* at NSWC, White Oak, MD (circa 1983); **b)** the much-improved third-generation version (2005 file photo).

We also replaced the bearing assembly for the steering axis (Figure 7-7), and upgraded the associated rubber drive belt and V-pulleys for the position-sensing potentiometer to a more robust timing belt configuration less susceptible to stretch and slippage errors.

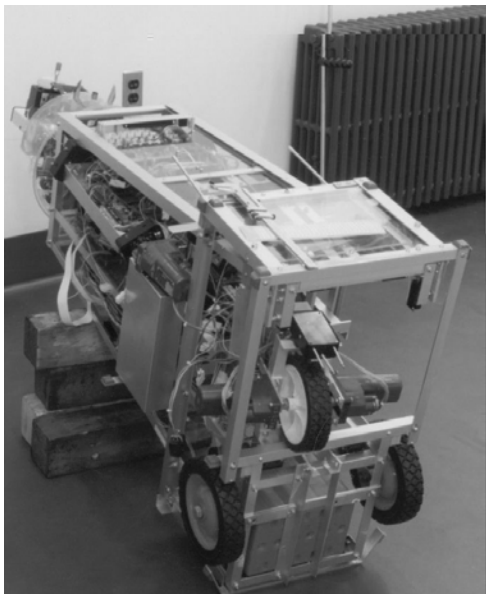


Figure 7-7. *ROBART I* rests horizontally while undergoing an upgrade to the steering-axis shaft-bearing assembly in this maintenance photo taken at the NSWC Robotics Lab, circa 1983.

All work with *ROBART I* ended in 1985, when the robot was shipped to Vancouver, BC, for display in the *Design 2000* exhibit at *EXPO '86*. This first-generation prototype is currently resident in the "museum" section of Building 624 of the Robotics Group at the Space and Naval Warfare Systems Center, San Diego, CA (Figure 7-8).

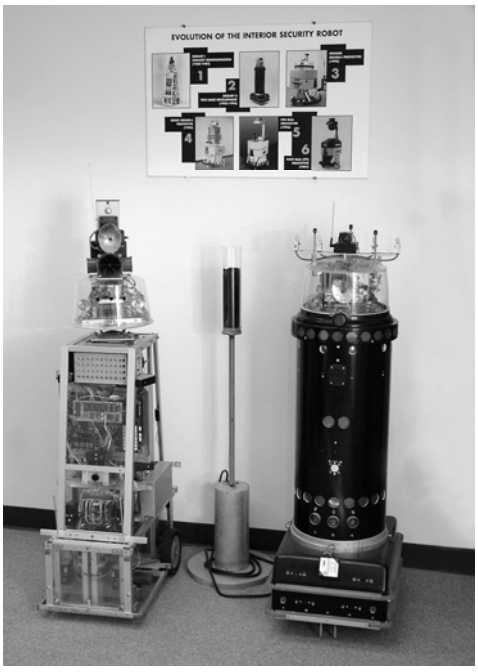


Figure 7-8. *ROBART I* and the second-generation *ROBART II* on display in Building 624.

7.5 References

Everett, H.R., "A Computer Controlled Sentry Robot," *Robotics Age,* pp. 22-27, March/April, 1982a.

Everett, H.R., *A Microprocessor Controlled Autonomous Sentry Robot,* Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982b.

Everett, H.R., "NAVSEA Integrated Robotics Program: Annual Report," FY-84, NAVSEA Technical Report No. 450-90G-TR-0002, Naval Sea Systems Command, Washington, DC, December, 1984.

Everett H.R., Gilbreath, G.A., Tran, T.T., Nieusma, J.M., *Modeling the Environment of a Mobile Security Robot,* NOSC Technical Document 1835, Naval Ocean Systems Center, San Diego, June, 1990.

Everett, H.R., and Pacis, E.P., "Towards a Warfighter's Associate: Eliminating the Operator Control Unit," SPIE Mobile Robots XVII, International Symposium on Optics East, Philadelphia, PA, 25-28 October, 2004.

Hawkins, Jeff, and Blakeslee, Sandra, *On Intelligence*, ISBN 0-8050-7456-2, Times Books, Henry Holt and Company, New York, NY, 2004.

Milgram P., and Kishino F., "A Taxonomy of Mixed Reality Visual Displays," *IEICE Transactions on Information Systems*, Special Issue on Networked Reality, Vol. E77-D, No. 12, December, 1994.

Webopedia, http://www.webopedia.com/TERM/a/artificial_intelligence.html, 2004.

Appendix A: Implementation Details

A-1 Interrupt Routines

The software structure for *ROBART I* could be broken down into three general areas. In addition to the main code that handled overall system control, there were two sections that dealt with interrupts: Non-maskable interrupts (NMIs) and maskable interrupt requests (IRQs). Coordination among these three areas was made possible by dedicating certain register locations in page zero (memory address locations $0000 - $00ff) for information transfer, and also by the fact that input/output devices could be accessed from all three areas, not just the main code.

There are two interrupt input lines to the 6502 microprocessor, each of which can temporarily halt the program under execution and cause a branch to a different location in memory. The processor would then execute the program stored at this new location, referred to as the *interrupt service routine*. This action by the processor is termed "responding to" or "servicing" an interrupt. Quite often, the processor branches to a specific location that contains the starting address of the interrupt service routine, and then branches again to that location. This concept of *vectored interrupts* allows the service routine to be located anywhere in memory, rather than restricted to a specific address. The specific address where the processor goes to fetch the service routine starting address is known as the interrupt vector location.

The *non-maskable interrupt* can override the lower priority *interrupt request*, and will occur whenever the NMI line (pin 6) is pulled low on the 6502. It is edge sensitive, occurring on the high-to-low transition, and cannot be internally masked by the processor. Similarly, an *interrupt request* occurs when the IRQ line (pin 4) goes low. Unlike the NMI, the IRQ is level sensitive, which means that the processor will be interrupted as long as pin 4 on the 6502 is held low. A second important difference is that *interrupt requests* can be temporarily disabled by setting a flag within the processor, called the *interrupt disable bit*. This bit can be set through software, and when set, causes all subsequent IRQs to be ignored. (The assembly-language command to set this bit is SEI, and it is cleared with the command CLI.)

It is important to note that this bit is set automatically by the 6502 when responding to an IRQ, and cleared automatically when returning from interrupt (RTI). The programmer has the option of changing its status as desired, either within the interrupt service routine or external to it. In any event, the condition causing the interrupt must first be dealt with before interrupts are re-enabled, or another interrupt will immediately occur, and an endless loop will result. The programmer has two options: (1) provide for eliminating the source of the interrupt through action initiated in the interrupt service routine, verify

elimination, and then return from interrupt, or (2) disable the device reading the interrupt (i.e., the hardware between the source and the 6502), then return from interrupt and attend to the cause. Once the condition has been cleared, the hardware that alerts the 6502 to an interrupt can be re-enabled for future use. Both methods were employed on *ROBART I.*

When a *non-maskable interrupt* occurs, the processor will complete the instruction currently under execution before recognizing the interrupt, and then store the contents of the *Program Counter* and the *Processor Status Register* on the stack. The processor then goes to a specific address in memory ($A67A) and fetches the starting address of the NMI service routine. In this manner, the 6502 can be halted, the required information saved on the stack to allow a return, and then vectored to a new set of instructions. On completion of these instructions, the processor can return and pick up where it left off, until interrupted again.

When an *interrupt request* occurs, provided the *interrupt disable bit* is clear, the current instruction is again completed, the *disable bit* is set, and the *Program Counter* and *Status Register* are saved on the stack. The processor then branches to a different address in memory ($A678) for the starting address of the IRQ service routine, and subsequently jumps to that indicated portion of memory for program execution.

After the interrupt has been serviced, whether NMI or IRG, the processor returns to the location of the next instruction to have been executed had the interrupt not occurred. This is termed a *return from interrupt* (RTI), and is accomplished by pulling the *Program Counter* and *Processor Status Register* off the stack, where they were previously stored. Execution then begins where the processor left off. Therefore, when routines are in progress where timing is critical, such as while waiting for a sonar echo, the Interrupt Disable bit should be set to prevent a lengthy interruption at a critical point, and then cleared upon completion of the routine. Any interrupts which may have occurred while the Disable Bit was set will be serviced as soon as the bit is cleared, as the IRQ line is level sensitive.

The NMI line cannot be ignored by the processor under any conditions, and so *NMI service routines* should be kept as short as possible where there is a chance they could interfere with other routines in progress. On *ROBART I*, the *non-maskable interrupt* was used to keep track of the time, incrementing the hours, minutes, and seconds registers as appropriate. Since it could not be masked out and had priority over IRQs, the accuracy of these registers was ensured, and service time kept to a minimum. The NMI interrupts were generated once each second by hardware circuitry located on the *Clock/Calendar Board*, as discussed in section 1 of the text.

The IRQ line connected to pin 4 of the 6502 could be pulled low by any of the three 6522 VIAs, or the 6532 RAM I/O Timer. Only one of these devices was used for interrupt handling on *ROBART I*, however, namely 6522-2. Each 6522 VIA had seven potential interrupt sources: four control lines, two timers, and one shift register. Only two sources were used: interrupts generated by Data Selector A (IRQ-A) are fed in on control line CA2 via AA Connector pin 4, and interrupts generated by Data Selector B

(IRQ-B) were fed in on control line CB-2 via AA Connector pin 5. These two sources were referred to as Interrupt Channel A and Interrupt Channel B, respectively. Data Selector C had no interrupt capability.

Each Data Selector could handle 16 different inputs, any one of which could generate an interrupt. Diode jumpers were installed on each selector input where an interrupt was required. Data Selector A used all 16 inputs to monitor the tactile bumpers/feelers and the near-infrared proximity detectors for collision avoidance. All inputs were jumpered to cause an interrupt, but the four proximity inputs could be disabled at the near-infrared driver board (so as to be ignored) by calling Subroutine *I/Rdis*.

Selector A interrupts could be collectively disabled, without affecting Selector A inputs, by calling subroutine *IRQAdis*, and enabled with subroutine *IRQAen*. Selector A was polled by subroutine *IRQA* once an interrupt was detected. Data Selector B was used to monitor internal circuitry check points, and not all inputs caused interrupts. Examples of those that did are the low-battery condition, analog-to-digital overflow, smoke alarm, fire alarm, and power distribution bus inputs. Selector B was polled by subroutine *IRQB* for branching to the appropriate service routine after returning from interrupt.

The *interrupt request service routine* first saved all primary registers on the stack. Next, the *Return* register was cleared, and the drive stopped with the current drive command stored for later use. Subroutine *IRQA* was called for polling Data Selector A, with Q (the IRQ index counter) initialized to start with input 4. The structure of subroutine IRQA was such that it would not return until all inputs on Selector A were low (see Figure A-1). Thus, the individual service subroutines for IRQA must clear the cause of the interrupt, or no return is possible.

Since Selector A read collision-related inputs generated by either tactile sensors or the close-in proximity detectors, the service subroutines called would command the drive wheel to move the platform away from the sensed object. When all inputs had been restored to a low state, subroutine *IRQA* returned to the *IRQ service routine*, which then called subroutine *IRQB*. Subroutine *IRQB* polled all inputs on Selector B in a similar fashion before returning, after which the *interrupt request routine* restored the old drive command, recalled the primary registers, and returned from interrupt.

Besides affecting drive motor responses within itself, the *IRQ service routine* also caused actions to be performed after return by setting certain registers before returning from interrupt. Register *Return*, for example, would cause a return from interrupt even though all inputs on Selector A were not low, if set by subroutine *IRQA*. Register *Homing*, if set, would cause *Skirt* to be activated during docking with the charger. A side impact at close range to the charging station would set register *Realign*, causing subroutine *Align* to be called when docking. These registers were predominantly addressed by Interrupt-Channel-A-related software concerned with collision avoidance. As further examples, register *Exit* could be set to ensure termination of a behavior routine in progress, and register *Next* could be set to pick the subsequent behavior. This

technique was used to process and react to alarm conditions generated by Interrupt Channel B.

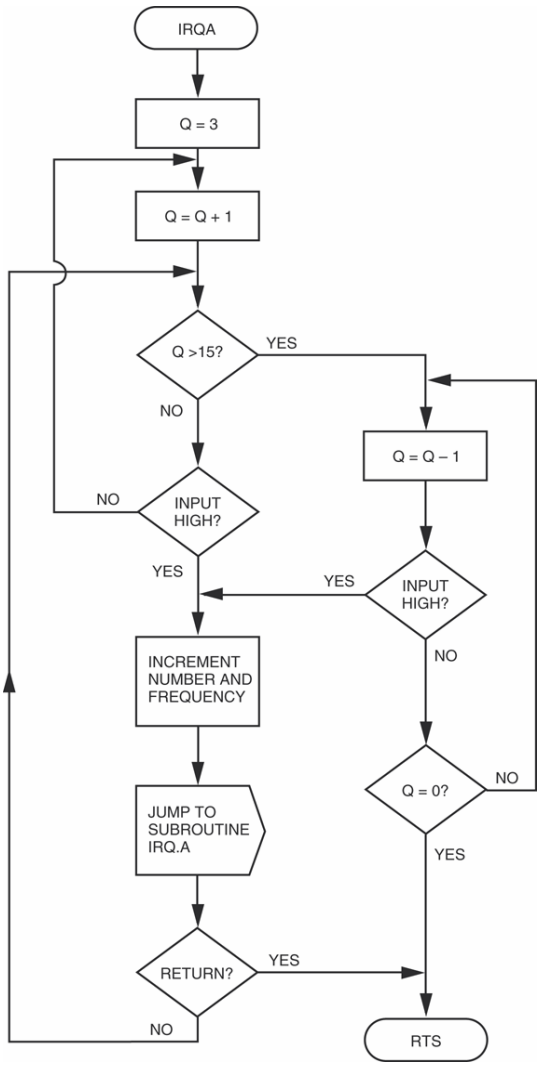


Figure A-1. Flowchart of the interrupt handling software associated with Data Selector A.

A-2 Proximity Detector Operation

The near-infrared proximity detector system consisted of one centrally located driver/detector board and several remotely mounted transmitter/receiver units, relocatable for optimum placement during evaluation of various collision-avoidance algorithms. The driver circuitry was built around two identical pulse generators, each producing a square-wave train of 15-microsecond pulses with a pulse-repetition period of 1.7 milliseconds. These pulses drove into saturation an NPN transistor that gated a XC-880-A high-power gallium-aluminum-arsenide LED emitter supplied in a T-1¾ package (Figure A-2).

A 47-mfd electrolytic and 10-ohm decoupling resistor were configured to supply an extremely heavy current flow (in excess of 2 amps) for the brief on-time, more than enough to destroy the LED under steady-state conditions. The result was an intense pulsed output in a narrow cone, both desirable properties for an object-detection system

of this type. The two pulse generators were alternately enabled by a 555 astable multivibrator at about a 1-Hertz rate, reducing power consumption by a factor of two, and eliminating pattern overlap of adjacent LEDs where desired. (There are certain cases where pattern overlap was intentionally used to shape and/or enhance the detection field of a sensor.)

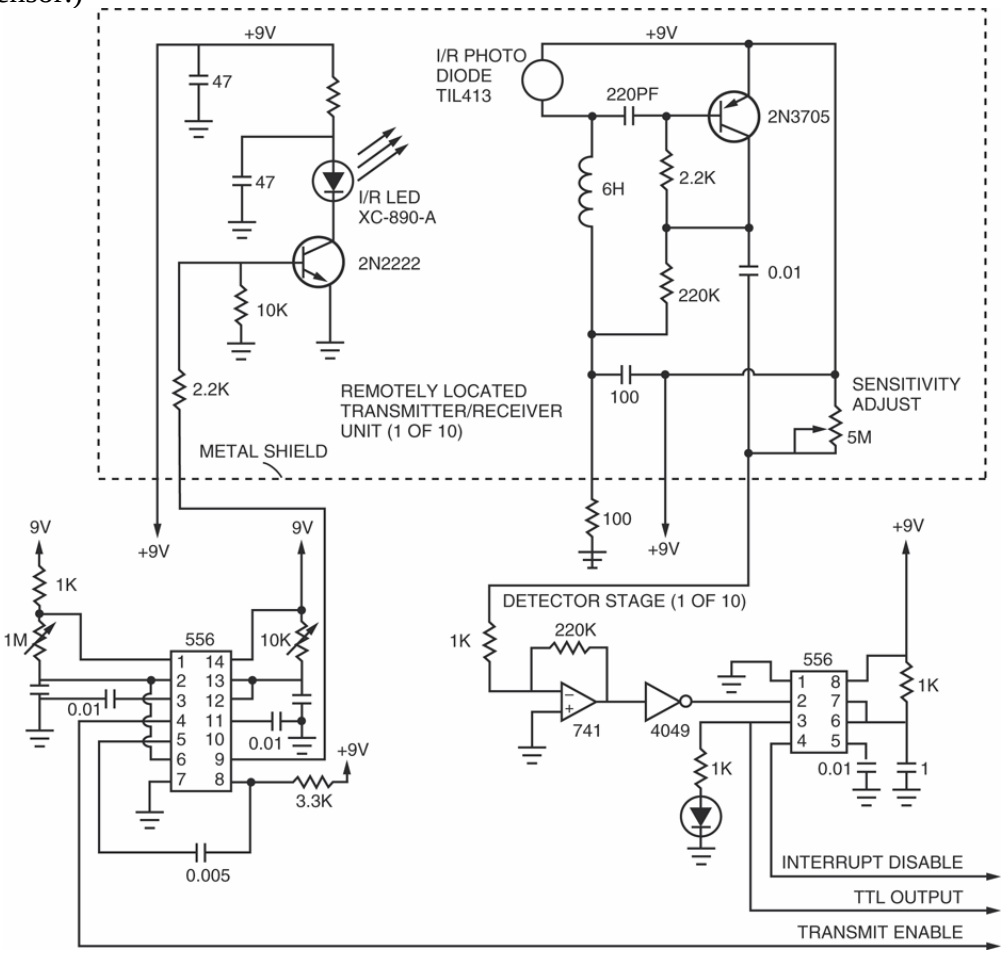


Figure A-2. Schematic diagram of the near-infrared proximity sensor.

The receivers associated with this system consisted of a TIL413 photodiode incorporating a built-in filter and lens system, with a cone-shaped detection field roughly 45 degrees around the lens axis. The output of this photodiode was amplified through a L/C differentiator network, and then fed to a 741 (1/2 458) op amp, which subsequently produced a positive spike for each burst of returned infrared energy detected. These pulses were inverted by a 4049, which also served as a threshold detector, and used to trigger a 555 monostable (1/2 556). The 555 served as a pulse stretcher, providing an output pulse of approximately 100 milliseconds, and illuminating a red LED for circuit monitoring and adjustment.

The receiver output generated an interrupt on IRQ Channel A and was then read by Data Selector A. Six receiver channels were provided, and all were commonly enabled or disabled as needed by Data Distributor A, output number 4. The circuitry was

powered up automatically with the Drive Relay Board by subroutine *Dri.on*. The receivers were subsequently enabled by subroutine *I/Ren*.

Position resolution of the detector was a function of receiver sensitivity, the photodiode field of view, and the irradiation pattern of the high-power LED emitter. Of these, the latter was the most significant determining factor, as the irradiation cone of the infrared LED was relatively narrow (40-degree angle between half-power points) with respect to the broader detection cone of the photodiode. The usable irradiation cone angle was experimentally determined to be roughly half that of the half-power angle, yielding fairly good angular resolution of object location for a low-cost proximity system.

Where desired, multiple emitters could be used to advantage to strengthen the irradiation field for greater range or sensitivity, and through careful placement of the LEDs, it was possible to shape the detection zone to best fit the application. The robot was little concerned with how tall an object was, but rather interested in horizontal resolution of its exact location. Emitters were therefore arranged in vertical columns to expand the detection field vertically while creating no horizontal overlap. This technique greatly increased the sensor's versatility at minimal additional cost, as long as the LED patterns remained within the photodiode field of view. A rough guideline to ensure reliability was found to be one LED irradiation pattern on either side of the photodetector, as shown in Figure A-3.

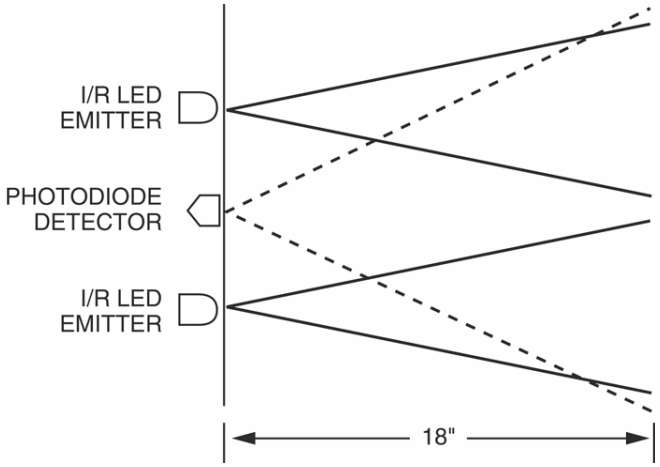


Figure A-3. Vertical coverage was increased through use of a second LED emitter, without adversely affecting the horizontal resolution of the proximity sensor.

A-3 Beacon-Tracking System

In *Scan Mode,* the head swept back and forth in search of the recharging beacon, controlled by the scan flip-flop on the *Optical Board* inside the robot's head. This flip-flop was set and reset by limit switches at both extremities of pan travel, causing the motor to reverse direction each time, making the head scan the other way. This mode was selected by subroutine *Scan*, which set the *Scan Enable* line high and the other two control lines low. When the *Scan Enable* line went low, the head was immobilized, provided the *Position Enable* line was also low. All three lines were set low by subroutine *Scanoff*.

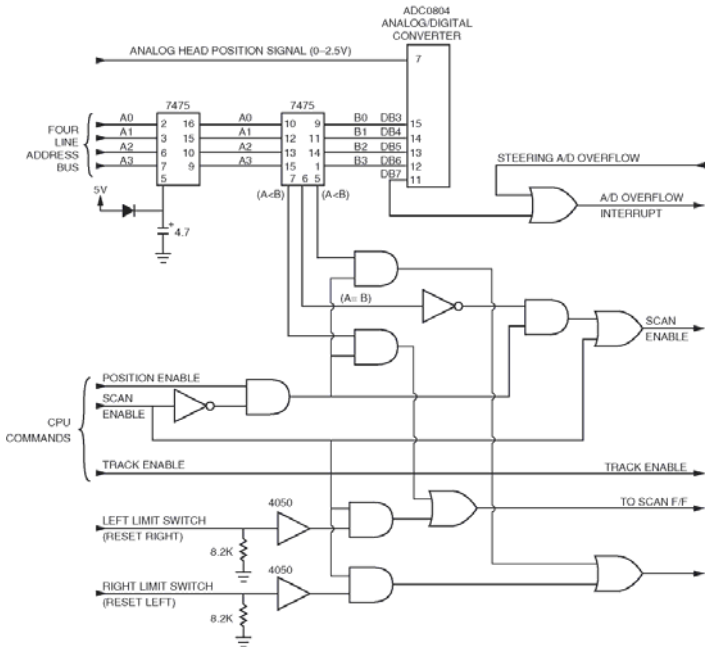


Figure A-4. In response to input commands from the computer, hardwired logic circuitry on *Interface Board Number 5* determined the control mode (i.e., *Position*, *Scan*, or *Track*) for the *Optical Board*.

If the *Position Enable* line was high, however, the head would seek the position stored by the CPU in the 7475 four-bit latch. This head positioning circuitry was automatically gated out whenever the *Scan Enable* line went high. The latch was loaded from the four-line I/O expansion bus and enabled by PB7 on 6522-2 (ORB2). These operations were automatically performed by subroutine *Latch*, which was passed the desired position command from the Y register. Thus, the head could be made to center itself after a scan operation, or to seek any of 16 fixed positions on command.

If *Track Enable* and *Scan Enable* were both high with *Position Enable* low, then the system functioned in *Track Mode*, and the head looked for, locked onto, and tracked the homing beacon situated on the recharging tower. The *Optical Board* provided three digital status outputs pertinent to the tracking process: (1) *Target Output*, (2) *Point Source Output*, and (3) *Range Output*. The *Target Output* went high if any of the three comparators associated with the photocell array acquired the beacon, while the *Point Source Output* reflected the status of the center photocell comparator only. The *Range Output* indicated relative distance from the homing beacon, as will be discussed later.

When the battery voltage dropped below the set point for more than 5 seconds, a flip-flop on the *Monitor Board* changed state and triggers an interrupt. The IRQ routine which handled the Channel B interrupts disabled the low-battery interrupt, and sets register *Next* to select the docking routine (see section 3). The transmitter that activated the homing beacon was enabled, along with the Automatic Scan and Track circuitry, while the *Position Enable* line was set low. The head would scan left and right, seeking a point source of light of sufficient intensity to trigger the photocell comparators. This action continued as long as *Scan Enable* and *Track Enable* were held high and no light

source was detected. If any of the three optical comparators indicated acquisition, the tracking inputs took over pan motor control.

The tracking inputs to the optical board circuitry came from the left and right photocells in the array. Their respective comparator outputs indicated a greater light intensity on either side of center, referenced to the center photocell output. The appropriate motor winding was energized so the head would turn to regain maximum intensity at the center photocell, thus tracking the source. (If the left and right photocells both showed intensities greater than center, their inputs were gated out and the head remained motionless.)

All this happened only if at least one of the photocell outputs was above the adjustable set-point provided by the *Background Light Bias Circuitry* on the *Optical Board*; otherwise, the system reverted to the scan mode and searched for a bright light source. Any comparator output signaling intensity above the set-point gated out the automatic scan, and the tracking inputs took over. When the array outputs indicated the head was correctly positioned (i.e., pointing at the source), the pan motor winding was de-energized.

The three collimating tubes limited the photocell fields-of-view to relatively small regions, and the beacon was situated at just the right height so as to be centered vertically within the detection zone when one of these tubes was pointed at the recharging station. This resulted in a relatively high signal-to-noise ratio (for household-sized rooms anyway) as the head scanned back and forth in search of the beacon, as shown by the stripchart recording of Figure A-5. The sharp peaks resulting from beacon acquisition readily stand out over the signal produced by ambient lighting during the sweep.

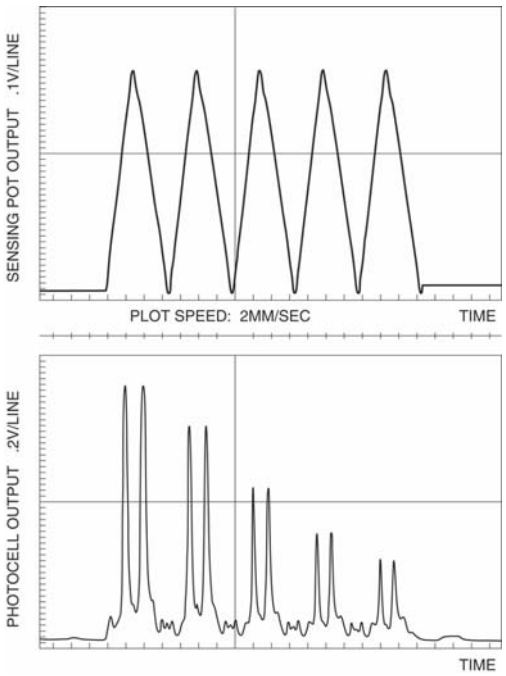


Figure A-5. Plot of the center photocell output for increasing range as a function of time (lower graph), as referenced to head scan position (upper graph).

The *Background Light Bias* circuitry was tasked with providing the comparators with a reference voltage above which photocell output was probably due to a point source of sufficient intensity to possibly be the homing beacon. The initial design provided for a bias potentiometer to manually set this reference level, which proved inadequate due to the inverse-square law, resulting in oversensitivity in close to the beacon. If the bias threshold was set low enough to allow detection of the beacon at long range, then the system saturated in close and all comparators went high when the pickup tubes were pointed in the general direction of the light. What was needed was a means of reducing the sensitivity as the robot approached the recharging station, from that needed for long-range detection to a level yielding good bearing resolution in close, where accuracy became critical.

The circuitry was therefore altered to provide two manually set reference points, one to allow sufficient sensitivity for long-range detection and a second with greatly reduced sensitivity for short-range use. This approach forced the tracking system to point the head directly at the light, instead of saturating in close, providing a more accurate bearing to the beacon for docking. A fourth comparator monitored the center photocell voltage output and initiated the changeover when its output signaled the robot was within 3 feet of the beacon, based solely on perceived light intensity (Figure A-6). This comparator output *(Range Output)* was also made available to the CPU, and used in many software routines to determine relative range (near or far) to the charging station. (A multi-channel analog-to-digital converter would have been a big help here.)

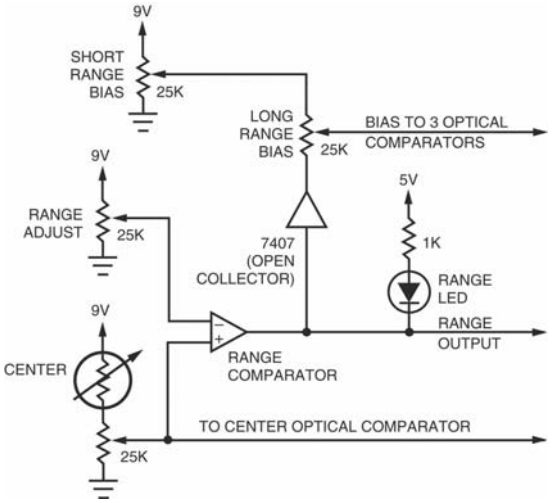


Figure A-6. The *Background Light Bias Circuitry* located on the *Optical Board* reset the comparator thresholds to prevent saturation in close to the beacon.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-01-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
12–2005	Final	15 March to 21 July 1999

4. TITLE AND SUBTITLE

ROBART I: IN RETROSPECT

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHORS

H. R. Everett

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SSC San Diego
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TD 3199

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSOR/MONITOR'S ACRONYM(S)

DARPA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This document discusses the development of ROBART I, one of the first behavior-based autonomous robots ever built.

15. SUBJECT TERMS

Mission Area: Robotics

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			H. R. Everett
U	U	U	UU	162	**19B. TELEPHONE NUMBER** *(Include area code)* (619) 553–3672

INITIAL DISTRIBUTION

360012	Patent Counsel	(1)
73511	S. Baxley	(1)
73512	Library	(2)
73513	Archive/Stock	(3)
71705	H. R. Everett	(100)

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (1)

SSC San Diego Liaison Office
C/O PEO-SCS
Arlington, VA 22202–4804 (1)

Center for Naval Analyses
Alexandria, VA 22311–1850 (1)

Government-Industry Data Exchange
Program Operations Center
Corona, CA 91718–8000 (1)